VRPH

1.0

Generated by Doxygen 1.6.1

Wed Apr 7 23:06:28 2010

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 ClarkeWright Class Reference

`#include <ClarkeWright.h>`

Public Member Functions

- ClarkeWright (int n)
- ~ClarkeWright ()
- bool Construct (class VRP *V, double lambda, bool use_neighbor_list)
- void CreateSavingsMatrix (class VRP *V, double lambda, bool use_neighbor_-list)

Public Attributes

- class VRPSavingsElement * s
- bool has_savings_matrix
- int savings_matrix_size

3.1.1 Detailed Description

Definition at line 21 of file ClarkeWright.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ClarkeWright::ClarkeWright (int *n*)

Sets up the data structure for the Clarke Wright savings algorithm including a matrix of size n(n-1)/2 for the savings values

Definition at line 16 of file ClarkeWright.cpp.

3.1.2.2 ClarkeWright::~ClarkeWright ()

Definition at line 29 of file ClarkeWright.cpp.

3.1.3 Member Function Documentation

3.1.3.1 bool ClarkeWright::Construct (class VRP * *V*, double *lambda*, bool *use_neighbor_list*)

This function constructs the routes via the Clarke Wright savings algorithm with the parameter lambda (see Yellow 19??): s_{ij}=d_{i0}+d_{0j}-lambda*d_{ij}.

Definition at line 122 of file ClarkeWright.cpp.

3.1.3.2 void ClarkeWright::CreateSavingsMatrix (class VRP * *V*, double *lambda*, bool *use_neighbor_list*)

This computes the savings matrix d[0,i]+d[0,j]-lambda*d[i,j] for all pairs (i,j) with i and j routed. Matrix is sorted and each element is of the form [val, i, j]

Definition at line 35 of file ClarkeWright.cpp.

3.1.4 Member Data Documentation

3.1.4.1 bool ClarkeWright::has_savings_matrix

Definition at line 29 of file ClarkeWright.h.

3.1.4.2 class VRPSavingsElement* ClarkeWright::s

Definition at line 27 of file ClarkeWright.h.

3.1.4.3 int ClarkeWright::savings_matrix_size

Definition at line 30 of file ClarkeWright.h.

The documentation for this class was generated from the following files:

- inc/ClarkeWright.h
- src/ClarkeWright.cpp

3.2 Concatenate Class Reference

`#include <Concatenate.h>`

Public Member Functions

- bool evaluate (class VRP *V, int i, int j, int criteria, VRPMove *M)
- bool move (VRP *V, int u, int i)

3.2.1 Detailed Description

Definition at line 15 of file Concatenate.h.

3.2.2 Member Function Documentation

3.2.2.1 bool Concatenate::evaluate (class VRP * *V*, int *i*, int *j*, int *criteria*, VRPMove * *M*)

This function takes route i_route and j_route and attempts to concatenate them together. Example: 1-i-a-b-c-...-x-1 and 1-aa-bb-...-zz-j-1 and merges them to form 1-aa-bb-...-zz-j-i-a-b-c-...-x-1 Note that the move_arguments i_route and j_route refer to the route numbers themselves and not the node numbers!

Definition at line 15 of file Concatenate.cpp.

3.2.2.2 bool Concatenate::move (VRP * *V*, int *u*, int *i*)

Attempts to merge the two routes i_route and j_route into a single route

Definition at line 85 of file Concatenate.cpp.

The documentation for this class was generated from the following files:

- inc/Concatenate.h
- src/Concatenate.cpp

3.3 CrossExchange Class Reference

`#include <CrossExchange.h>`

Public Member Functions

- bool route_search (class VRP *V, int r1, int r2, int criteria)

Private Member Functions

- bool evaluate (class VRP *V, int i1, int i2, int k1, int k2, int j1, int j2, int l1, int l2, int criteria, VRPMove *M)
- bool move (class VRP *V, VRPMove *M)

3.3.1 Detailed Description

Definition at line 15 of file CrossExchange.h.

3.3.2 Member Function Documentation

3.3.2.1 bool CrossExchange::evaluate (class VRP * *V*, int *i1*, int *i2*, int *k1*, int *k2*, int *j1*, int *j2*, int *l1*, int *l2*, int *criteria*, VRPMove * *M*) [private]

Evaluate the move of removing the edges i1-i2 and k1-k2 in one route and j1-j2 and l1-l2 in another route and replacing these edges with i1-j2, j1-i2, k1-l2, and l1-k2

Definition at line 238 of file CrossExchange.cpp.

3.3.2.2 bool CrossExchange::move (class VRP * *V*, VRPMove * *M*) [private]

Make the move of removing the edges i1-i2 and k1-k2 in one route and j1-j2 and l1-l2 in another route and replacing these edges with i1-j2, j1-i2, k1-l2, and l1-k2. See

Definition at line 375 of file CrossExchange.cpp.

3.3.2.3 bool CrossExchange::route_search (class VRP * *V*, int *r1*, int *r2*, int *criteria*)

Attempts to find a cross exchange move between routes r1 and r2. Edges i1-i2 and k1-k2 in route r1, and edges j1-j2 and l1-l2 in route r2.

Definition at line 16 of file CrossExchange.cpp.

The documentation for this class was generated from the following files:

- inc/CrossExchange.h
- src/CrossExchange.cpp

3.4 double_int Struct Reference

`#include <VRPUtils.h>`

Public Attributes

- double d
- int k

3.4.1 Detailed Description

Definition at line 52 of file VRPUtils.h.

3.4.2 Member Data Documentation

3.4.2.1 double double_int::d

Definition at line 54 of file VRPUtils.h.

3.4.2.2 int double_int::k

Definition at line 55 of file VRPUtils.h.

The documentation for this struct was generated from the following file:

- inc/VRPUtils.h

3.5 Flip Class Reference

`#include <Flip.h>`

Public Member Functions

- bool evaluate (class VRP *V, int i, int j, VRPMove *M)
- bool move (VRP *V, int u, int i)

3.5.1 Detailed Description

Definition at line 16 of file Flip.h.

3.5.2 Member Function Documentation

3.5.2.1 bool Flip::evaluate (class VRP * *V*, int *i*, int *j*, VRPMove * *M*)

Evaluates the move of reversing a portion of a route in between nodes start and end. Example: 0-a-b-start-d-e-f-g-h-end-x-y-z-0 becomes 0-a-b-start-h-g-f-e-d-end-x-y-z-0. If the move is feasible, the information regarding the move is stored in the VRPMove data structure M. start_point must be before end_point in the current route orientation.

Definition at line 16 of file Flip.cpp.

3.5.2.2 bool Flip::move (VRP * *V*, int *u*, int *i*)

This reverses the portion of the route between start_point and end_point if the proposed move is feasible. Returns true and makes all relevant solution modifications if the move is made and false otherwise.

Definition at line 94 of file Flip.cpp.

The documentation for this class was generated from the following files:

- inc/Flip.h
- src/Flip.cpp

3.6 htable_entry Struct Reference

`#include <VRPUtils.h>`

Public Attributes

- int num_vals
- int hash_val_2 [NUM_ENTRIES]
- int tot
- double length [NUM_ENTRIES]

3.6.1 Detailed Description

Definition at line 30 of file VRPUtils.h.

3.6.2 Member Data Documentation

3.6.2.1 int htable_entry::hash_val_2[NUM_ENTRIES]

Definition at line 41 of file VRPUtils.h.

3.6.2.2 double htable_entry::length[NUM_ENTRIES]

Definition at line 43 of file VRPUtils.h.

3.6.2.3 int htable_entry::num_vals

Each entry in the hash table will contain an array of num_vals valid entries in hval[]. Each entry is produced by hashing using SALT_2. The length array contains the lengths of the routes in this position in the hash table.

Definition at line 40 of file VRPUtils.h.

3.6.2.4 int htable_entry::tot

Definition at line 42 of file VRPUtils.h.

The documentation for this struct was generated from the following file:

- inc/VRPUtils.h

3.7 int_int Struct Reference

`#include <VRPUtils.h>`

Public Attributes

- int i
- int j

3.7.1 Detailed Description

Definition at line 47 of file VRPUtils.h.

3.7.2 Member Data Documentation

3.7.2.1 int int_int::i

Definition at line 49 of file VRPUtils.h.

3.7.2.2 int int_int::j

Definition at line 50 of file VRPUtils.h.

The documentation for this struct was generated from the following file:

- inc/VRPUtils.h

3.8 MoveString Class Reference

`#include <MoveString.h>`

Public Member Functions

- bool evaluate (class VRP *V, int a, int b, int u, int v, VRPMove *M)
- bool move (class VRP *V, int a, int b, int u, int v)

3.8.1 Detailed Description

Definition at line 16 of file MoveString.h.

3.8.2 Member Function Documentation

3.8.2.1 bool MoveString::evaluate (class VRP * *V*, int *a*, int *b*, int *u*, int *v*, VRPMove * *M*)

Evaluates the move of taking the string between u and v (i.e. t-u-j-k-l-m-v-w) and inserting between a and b (assumed to currently be an existing edge), yielding t-w & a-u-j-k-l-m-b

Definition at line 14 of file MoveString.cpp.

3.8.2.2 bool MoveString::move (class VRP * *V*, int *a*, int *b*, int *u*, int *v*)

Takes the string of nodes between u and v (inclusive) and places it between a and b.

Definition at line 179 of file MoveString.cpp.

The documentation for this class was generated from the following files:

- inc/MoveString.h
- src/MoveString.cpp

3.9 OnePointMove Class Reference

`#include <OnePointMove.h>`

Public Member Functions

- bool search (class VRP *V, int i, int rules)
- bool route_search (class VRP *V, int r1, int r2, int rules)

Private Member Functions

- bool evaluate (class VRP *V, int j, int b, int rules, VRPMove *M)
- bool move (class VRP *V, VRPMove *M)

3.9.1 Detailed Description

Definition at line 16 of file OnePointMove.h.

3.9.2 Member Function Documentation

3.9.2.1 bool OnePointMove::evaluate (class VRP * *V*, int *j*, int *b*, int *rules*, VRPMove * *M*) [private]

This function evaluates the move of inserting j either before or after node b and places the best savings found in the VRPMove struct M if the move is feasible and returns false if no feasible move is found, true otherwise.

Definition at line 268 of file OnePointMove.cpp.

3.9.2.2 bool OnePointMove::move (class VRP * *V*, VRPMove * *M*) [private]

Makes the one point move determined by the VRPMove M.

Definition at line 496 of file OnePointMove.cpp.

3.9.2.3 bool OnePointMove::route_search (class VRP * *V*, int *r1*, int *r2*, int *rules*)

Searches for a one point move where a node from route r1 is moved into route r2.

Definition at line 152 of file OnePointMove.cpp.

3.9.2.4 bool OnePointMove::search (class VRP * *V*, int *i*, int *rules*)

Attempts to find an appropriate one point move involving node j using the specified rules. If acceptable move is found, the move is made and function returns true. Returns false if no move is found.

Definition at line 16 of file OnePointMove.cpp.

The documentation for this class was generated from the following files:

- inc/OnePointMove.h
- src/OnePointMove.cpp

3.10 OrOpt Class Reference

`#include <OrOpt.h>`

Public Member Functions

- bool search (class VRP *V, int i, int j, int rules)
- bool route_search (class VRP *V, int r1, int r2, int k, int rules)

Private Member Functions

- bool evaluate (class VRP *V, int a, int len, int c, int d, int rules, VRPMove *M)
- bool move (class VRP *V, VRPMove *M)

3.10.1 Detailed Description

Definition at line 17 of file OrOpt.h.

3.10.2 Member Function Documentation

3.10.2.1 bool OrOpt::evaluate (class VRP * *V*, int *a*, int *len*, int *c*, int *d*, int *rules*, VRPMove * *M*) [private]

Evaluates the move of taking the string of length len beginning at a and inserting it between c and d subject to the provided rules

Definition at line 297 of file OrOpt.cpp.

3.10.2.2 bool OrOpt::move (class VRP * *V*, VRPMove * *M*) [private]

Modifies all solution information by taking the string of length len at and inserting between c and d if it meets the rules

Definition at line 388 of file OrOpt.cpp.

3.10.2.3 bool OrOpt::route_search (class VRP * *V*, int *r1*, int *r2*, int *k*, int *rules*)

Searches for the best OrOpt move where we take a string of length len from route r1 and try to move the string into route r2 (and vice versa) subject to the provided rules

Definition at line 213 of file OrOpt.cpp.

3.10.2.4 bool OrOpt::search (class VRP * *V*, int *i*, int *j*, int *rules*)

Looks for string insertions of length len beginning at a that meet the provided rules. Makes move if one is found.

Definition at line 16 of file OrOpt.cpp.

The documentation for this class was generated from the following files:

- inc/OrOpt.h
- src/OrOpt.cpp

3.11 Postsert Class Reference

`#include <Postsert.h>`

Public Member Functions

- bool evaluate (class VRP *V, int i, int j, VRPMove *M)
- bool move (VRP *V, int u, int i)

3.11.1 Detailed Description

Definition at line 16 of file Postsert.h.

3.11.2 Member Function Documentation

3.11.2.1 bool Postsert::evaluate (class VRP * *V*, int *i*, int *j*, VRPMove * *M*)

Evaluates the move of placing u AFTER node i in whatever route node i is currently in. If a move is found, the relevant solution modification information is placed in the VRPMove M

Definition at line 15 of file Postsert.cpp.

3.11.2.2 bool Postsert::move (VRP * *V*, int *u*, int *i*)

This function inserts node number u AFTER node i in whatever route node i is currently in and modifies all relevant solution information.

Definition at line 184 of file Postsert.cpp.

The documentation for this class was generated from the following files:

- inc/Postsert.h
- src/Postsert.cpp

3.12 Presert Class Reference

`#include <Presert.h>`

Public Member Functions

- bool evaluate (class VRP *V, int i, int j, VRPMove *M)
- bool move (VRP *V, int u, int i)

3.12.1 Detailed Description

Definition at line 17 of file Presert.h.

3.12.2 Member Function Documentation

3.12.2.1 bool Presert::evaluate (class VRP * *V*, int *i*, int *j*, VRPMove * *M*)

This function evaluates the move where we insert node u BEFORE node i in whatever route node i is currently in. If the move is feasible, the relevant solution information is placed in the VRPMove M.

Definition at line 15 of file Presert.cpp.

3.12.2.2 bool Presert::move (VRP * *V*, int *u*, int *i*)

This function inserts node number u BEFORE node i in whatever route node i is currently in if the move is feasible.

Definition at line 167 of file Presert.cpp.

The documentation for this class was generated from the following files:

- inc/Presert.h
- src/Presert.cpp

3.13 Swap Class Reference

`#include <Swap.h>`

Public Member Functions

- bool evaluate (class VRP *V, int i, int j, VRPMove *M)
- bool move (VRP *V, int u, int i)

3.13.1 Detailed Description

Definition at line 16 of file Swap.h.

3.13.2 Member Function Documentation

3.13.2.1 bool Swap::evaluate (class VRP * *V*, int *i*, int *j*, VRPMove * *M*)

Evaluates the move of swapping the positions of nodes u and i in the current For example, Current situation: t-u-v and h-i-j New situation: t-i-v and h-u-k

!!

Definition at line 16 of file Swap.cpp.

3.13.2.2 bool Swap::move (VRP * *V*, int *u*, int *i*)

This modifies the current solution information by swapping the positions of u and i in the current configuration if the proposed move meets the rules. Returns true if the move succeeds and false otherwise.

Definition at line 205 of file Swap.cpp.

The documentation for this class was generated from the following files:

- inc/Swap.h
- src/Swap.cpp

3.14 SwapEnds Class Reference

`#include <SwapEnds.h>`

Public Member Functions

- bool evaluate (class VRP *V, int i, int j, VRPMove *M)
- bool move (VRP *V, int u, int i)

3.14.1 Detailed Description

Definition at line 16 of file SwapEnds.h.

3.14.2 Member Function Documentation

3.14.2.1 bool SwapEnds::evaluate (class VRP * *V*, int *i*, int *j*, VRPMove * *M*)

This function takes the routes containing nodes a and v and evaluates the swapping of the ends of the routes following a and v respectively, subject to the provided rules. Example: VRPH_DEPOT-i-a-j-k-l-VRPH_DEPOT and VRPH_DEPOT-t-u-v-x-y-z-VRPH_DEPOT becomes VRPH_DEPOT-i-a-x-y-z-VRPH_DEPOT and VRPH_DEPOT-t-u-v-j-k-l-VRPH_DEPOT

Example: (a & v input): VRPH_DEPOT-i-a-b-j-k-l-VRPH_DEPOT and VRPH_DEPOT-t-u-v-w-x-y-z-VRPH_DEPOT becomes VRPH_DEPOT-i-a-w-x-y-z-VRPH_DEPOT and VRPH_DEPOT-t-u-v-b-j-k-l-VRPH_DEPOT

Definition at line 15 of file SwapEnds.cpp.

3.14.2.2 bool SwapEnds::move (VRP * *V*, int *u*, int *i*)

This function takes the routes corresponding to nodes a and v and swaps the ends of these routes following a and v respectively. Example: VRPH_DEPOT-i-a-j-k-l-VRPH_DEPOT and VRPH_DEPOT-t-u-v-x-y-z-VRPH_DEPOT becomes VRPH_DEPOT-i-a-x-y-z-VRPH_DEPOT and VRPH_DEPOT-t-u-v-j-k-l-VRPH_DEPOT. If the proposed move is feasible, all solution modifications are made, and the function returns true. Returns false if the move is infeasible.

Definition at line 99 of file SwapEnds.cpp.

The documentation for this class was generated from the following files:

- inc/SwapEnds.h
- src/SwapEnds.cpp

3.15 Sweep Class Reference

`#include <Sweep.h>`

Public Member Functions

- Sweep ()
- bool Construct (class VRP *V)

3.15.1 Detailed Description

Definition at line 20 of file Sweep.h.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 Sweep::Sweep ()

Definition at line 21 of file Sweep.cpp.

3.15.3 Member Function Documentation

3.15.3.1 bool Sweep::Construct (class VRP * *V*)

Constructs an initial VRP solution by the simple sweep method. Start by picking a random node and then sweep counterclockwise and add nodes until we reach vehicle capacity or max route length. Improve after every imp_interval additions by running the provided heuristics (VRPH_DOWNHILL only).

Definition at line 26 of file Sweep.cpp.

The documentation for this class was generated from the following files:

- inc/Sweep.h
- src/Sweep.cpp

3.16 sweep_node Struct Reference

Public Attributes

- double theta
- int index

3.16.1 Detailed Description

Definition at line 15 of file Sweep.cpp.

3.16.2 Member Data Documentation

3.16.2.1 int sweep_node::index

Definition at line 18 of file Sweep.cpp.

3.16.2.2 double sweep_node::theta

Definition at line 17 of file Sweep.cpp.

The documentation for this struct was generated from the following file:

- src/Sweep.cpp

3.17 ThreeOpt Class Reference

`#include <ThreeOpt.h>`

Public Member Functions

- bool route_search (class VRP *V, int r, int criteria)

Private Member Functions

- bool evaluate (class VRP *V, int a, int b, int c, int d, int e, int f, int criteria, VRPMove *M)
- bool move (class VRP *V, VRPMove *M)

3.17.1 Detailed Description

Definition at line 16 of file ThreeOpt.h.

3.17.2 Member Function Documentation

3.17.2.1 bool ThreeOpt::evaluate (class VRP * *V*, int *a*, int *b*, int *c*, int *d*, int *e*, int *f*, int *criteria*, VRPMove * *M*) [private]

Evaluates the Three-Opt move involving the directed edges ab, cd, and ef, subject to the given rules. The function finds the most cost effective of the possible moves and stores the relevant data in the VRPMove M and returns true. If no satisfactory move is found, the function returns false.

Definition at line 214 of file ThreeOpt.cpp.

3.17.2.2 bool ThreeOpt::move (class VRP * *V*, VRPMove * *M*) [private]

This function makes the actual solution modification involving the Three-Opt move with the edges V->d[a][b], V->d[c][d], and V->d[e][f].

!!!!

!!!!

!!

Definition at line 305 of file ThreeOpt.cpp.

3.17.2.3 bool ThreeOpt::route_search (class VRP * *V*, int *r*, int *criteria*)

Searches for a Three-Opt move in route r. If a satisfactory move is found, then the move is made. If no move is found, false is returned.

Definition at line 16 of file ThreeOpt.cpp.

The documentation for this class was generated from the following files:

- inc/ThreeOpt.h
- src/ThreeOpt.cpp

3.18 ThreePointMove Class Reference

`#include <ThreePointMove.h>`

Public Member Functions

- bool search (class VRP *V, int b, int criteria)
- bool route_search (class VRP *V, int r1, int r2, int criteria)

Private Member Functions

- bool evaluate (class VRP *V, int a, int j, int k, int criteria, VRPMove *M)
- bool move (class VRP *V, VRPMove *M)

3.18.1 Detailed Description

Definition at line 16 of file ThreePointMove.h.

3.18.2 Member Function Documentation

3.18.2.1 bool ThreePointMove::evaluate (class VRP * *V*, int *a*, int *j*, int *k*, int *criteria*, VRPMove * *M*) [private]

Evaluates the move of exchanging node b with the position of edge i-j. subject to the given rules. Details of move placed in the VRPMove M if it meets rules.

Definition at line 309 of file ThreePointMove.cpp.

3.18.2.2 bool ThreePointMove::move (class VRP * *V*, VRPMove * *M*) [private]

Makes the ThreePointMove specified by M.

Definition at line 464 of file ThreePointMove.cpp.

3.18.2.3 bool ThreePointMove::route_search (class VRP * *V*, int *r1*, int *r2*, int *criteria*)

Searches for all ThreePointMoves involving two nodes from route r1 and one node from route r2.

Definition at line 185 of file ThreePointMove.cpp.

3.18.2.4 bool ThreePointMove::search (class VRP * *V*, int *b*, int *criteria*)

Searches for ThreePointMoves involving node b. In this move, the position of node b is exchanged with two other nodes in an existing edge.

Definition at line 15 of file ThreePointMove.cpp.

The documentation for this class was generated from the following files:

- inc/ThreePointMove.h
- src/ThreePointMove.cpp

3.19 TwoOpt Class Reference

`#include <TwoOpt.h>`

Public Member Functions

- bool search (class VRP *V, int i, int criteria)
- bool route_search (class VRP *V, int r1, int r2, int criteria)

Private Member Functions

- bool evaluate (class VRP *V, int a, int b, int c, int d, int criteria, VRPMove *M)
- bool move (class VRP *V, VRPMove *M)

3.19.1 Detailed Description

Definition at line 16 of file TwoOpt.h.

3.19.2 Member Function Documentation

3.19.2.1 bool TwoOpt::evaluate (class VRP * *V*, int *a*, int *b*, int *c*, int *d*, int *criteria*, VRPMove * *M*) [private]

Considers the Two-Opt move involving the edges a-b and c-d and the provided rules. If the move meets the rules, then the relevant changes to the solution are stored in the VRPMove M and the function returns true. Returns false otherwise.

Definition at line 580 of file TwoOpt.cpp.

3.19.2.2 bool TwoOpt::move (class VRP * *V*, VRPMove * *M*) [private]

Makes the actual solution modification implied by the Two-Opt move involving edges a-b and c-d. Handles both intraroute and interroute moves

Definition at line 890 of file TwoOpt.cpp.

3.19.2.3 bool TwoOpt::route_search (class VRP * *V*, int *r1*, int *r2*, int *criteria*)

Searches for all two opt moves involving an edge from r1 and an edge from r2.

Definition at line 478 of file TwoOpt.cpp.

3.19.2.4 bool TwoOpt::search (class VRP * *V*, int *i*, int *criteria*)

Attempts to find the best Two-Opt move involving node b using the specified rules. If acceptable move is found, the move is made and all relevant solution modifications are made.

Definition at line 16 of file TwoOpt.cpp.

The documentation for this class was generated from the following files:

- inc/TwoOpt.h
- src/TwoOpt.cpp

3.20 TwoPointMove Class Reference

`#include <TwoPointMove.h>`

Public Member Functions

- bool search (class VRP ∗V, int i, int rules)
- bool route_search (class VRP ∗V, int r1, int r2, int rules)

Private Member Functions

- bool evaluate (class VRP ∗V, int i, int j, int rules, VRPMove ∗M)
- bool move (class VRP ∗V, VRPMove ∗M)

3.20.1 Detailed Description

Definition at line 17 of file TwoPointMove.h.

3.20.2 Member Function Documentation

3.20.2.1 bool TwoPointMove::evaluate (class VRP ∗ *V*, int *i*, int *j*, int *rules*, VRPMove ∗ *M*) [private]

This function evaluates the move of swapping the positions of j and b in the current solution. If a satisfactory move is found subject to the provided rules, then the solution modification data is placed in the VRPMove M and the function returns true. Returns false otherwise.

Definition at line 256 of file TwoPointMove.cpp.

3.20.2.2 bool TwoPointMove::move (class VRP ∗ *V*, VRPMove ∗ *M*) [private]

Performs the actual solution modification given by the move M.

Definition at line 320 of file TwoPointMove.cpp.

3.20.2.3 bool TwoPointMove::route_search (class VRP ∗ *V*, int *r1*, int *r2*, int *rules*)

Searches for all TPM moves involving a node from route r1 and the other from route r2.

Definition at line 162 of file TwoPointMove.cpp.

3.20.2.4 bool TwoPointMove::search (class VRP * *V*, int *i*, int *rules*)

Attempts to find the best Two-Point move involving node j using the specified search space, and rules. If an acceptable move is found, then the move is made and all relevant solution modifications are made.

Definition at line 17 of file TwoPointMove.cpp.

The documentation for this class was generated from the following files:

- inc/TwoPointMove.h
- src/TwoPointMove.cpp

3.21 VRP Class Reference

`#include <VRP.h>`

Public Member Functions

- VRP (int n)
- VRP (int n, int ndays)
- ~VRP ()
- void read_TSPLIB_file (const char *infile)
- void write_TSPLIB_file (const char *outfile)
- void show_next_array ()
- void show_pred_array ()
- bool verify_routes (const char *message)
- bool check_fixed_edges (const char *message)
- void create_pred_array ()
- void print_stats ()
- void write_solution_file (const char *filename)
- void write_solutions (int num_sols, const char *filename)
- void write_tex_file (const char *filename)
- void read_solution_file (const char *filename)
- int read_fixed_edges (const char *filename)
- void export_solution_buff (int *sol_buff)
- void import_solution_buff (int *sol_buff)
- void export_canonical_solution_buff (int *sol_buff)
- void show_routes ()
- void show_route (int k)
- void summary ()
- void reset ()
- bool plot (const char *filename, int options, int orientation)
- bool plot (const char *filename)
- bool plot_route (int r, const char *filename)
- bool clone (VRP *W)
- double RTR_solve (int heuristics, int intensity, int max_stuck, int num_perturbs, double dev, int nlist_size, int perturb_type, int accept_type, bool verbose)
- double SA_solve (int heuristics, double start_temp, double cool_ratio, int iters_per_loop, int num_loops, int nlist_size, bool verbose)
- void set_daily_demands (int day)
- void set_daily_service_times (int day)
- bool create_default_routes ()
- bool create_default_routes (int day)
- bool eject_node (int k)

- int inject_set (int num, int *nodelist, int rules, int attempts)
- void eject_neighborhood (int j, int num, int *nodelist)
- void refresh_routes ()
- void normalize_route_numbers ()
- void update_route (int j, VRPRoute *R)
- void clean_route (int r, int heuristics)
- double split (double p)
- int split_routes (double p, int **ejected_routes, double *t)
- void add_route (int *route_buff)
- void append_route (int *sol_buff, int *route_buff)
- int intersect_solutions (int *new_sol, int **routes, int *sol1, int *sol2, int min_routes)
- int find_common_routes (int *sol1, int *sol2, int *route_nums)
- void list_fixed_edges (int *fixed_list)
- void unfix_all ()
- void fix_edge (int start, int end)
- void unfix_edge (int start, int end)
- void fix_string (int *node_string, int k)
- int get_num_nodes ()
- double get_total_route_length ()
- double get_total_service_time ()
- double get_best_sol_buff (int *sol_buff)
- double get_best_total_route_length ()
- int get_total_number_of_routes ()
- int get_num_original_nodes ()
- int get_num_days ()
- double get_best_known ()
- void set_best_total_route_length (double val)
- int get_max_veh_capacity ()
- double get_max_route_length ()
- void create_distance_matrix (int type)
- void create_neighbor_lists (int nsize)
- bool perturb ()
- bool eject_route (int r, int *route_buff)
- bool inject_node (int j)
- void reverse_route (int i)

Public Attributes

- char name [VRPH_STRING_SIZE]
- VRPSolutionWarehouse * solution_wh
- VRPRouteWarehouse * route_wh
- int num_evaluations [NUM_HEURISTICS]
- int num_moves [NUM_HEURISTICS]

Private Member Functions

- bool create_search_neighborhood (int j, int rules)
- bool check_tabu_status (VRPMove *M, int *old_sol)
- bool before (int a, int b)
- bool check_feasibility (VRPViolation *VV)
- bool is_feasible (VRPMove *M, int rules)
- bool postsert_dummy (int i)
- bool presert_dummy (int i)
- bool remove_dummy ()
- bool osman_insert (int k, double alpha)
- int osman_perturb (int num, double alpha)
- bool insert_node (int j, int i, int k)
- void perturb_locations (double c)
- void find_cheapest_insertion (int j, int *edge, double *costs, int rules)
- double insertion_cost (int u, int a, int b)
- double ejection_cost (int u)
- void update (VRPMove *M)
- void compute_route_center (int r)
- void find_neighboring_routes ()
- void capture_best_solution ()
- void update_solution_wh ()
- bool get_segment_info (int a, int b, struct VRPSegment *S)
- double get_distance_between (int a, int b)
- int get_string_end (int a, int len)
- int count_num_routes ()
- void update_arrival_times ()
- bool check_move (VRPMove *M, int rules)
- bool check_savings (VRPMove *M, int rules)

Private Attributes

- int num_nodes
- double total_route_length
- double total_service_time
- int * best_sol_buff
- double best_total_route_length
- int total_number_of_routes
- int num_original_nodes
- double best_known
- int num_days
- int problem_type

- int total_demand
- int max_veh_capacity
- int orig_max_veh_capacity
- double max_route_length
- double min_route_length
- double orig_max_route_length
- int min_vehicles
- bool has_service_times
- double fixed_service_time
- int edge_weight_type
- int coord_type
- int display_type
- int edge_weight_format
- int matrix_size
- double balance_parameter
- int dummy_index
- int neighbor_list_size
- double temperature
- double cooling_ratio
- bool symmetric
- bool can_display
- double ** d
- bool ** fixed
- class VRPNode * nodes
- bool depot_normalized
- bool forbid_tiny_moves
- int search_size
- int * search_space
- int * next_array
- int * pred_array
- int * route_num
- bool * routed
- class VRPRoute * route
- class VRPTabuList * tabu_list
- double record
- double deviation
- double min_theta
- double max_theta
- int * current_sol_buff
- class VRPViolation violation

Friends

- class OnePointMove
- class TwoPointMove
- class ThreePointMove
- class TwoOpt
- class ThreeOpt
- class OrOpt
- class CrossExchange
- class Postsert
- class Presert
- class MoveString
- class Swap
- class SwapEnds
- class Concatenate
- class Flip
- class ClarkeWright
- class Sweep

3.21.1 Detailed Description

Definition at line 17 of file VRP.h.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 VRP::VRP (int *n*)

Constructor for an n-node problem.

Definition at line 24 of file VRP.cpp.

3.21.2.2 VRP::VRP (int *n*, int *ndays*)

Constructor for an n-node, ndays-day problem.

Definition at line 107 of file VRP.cpp.

3.21.2.3 VRP::~VRP ()

Destructor for the VRP.

Definition at line 194 of file VRP.cpp.

3.21.3 Member Function Documentation

3.21.3.1 void VRP::add_route (int * *route_buff*)

Adds the route in the provided buffer to the solution. The new route should not have any nodes in common with the existing solution! The route_buff[] should be terminated with a -1.

Definition at line 3747 of file VRP.cpp.

3.21.3.2 void VRP::append_route (int * *sol_buff*, int * *route_buff*)

Appends the single route contained in route_buff[] (which ends in a -1) to the solution buffer sol_buff, updating the first entry in sol_buff which is the # of nodes in the solution. Does NOT import the resulting solution and assumes that route_buff and sol_buff are disjoint.

Definition at line 3790 of file VRP.cpp.

3.21.3.3 bool VRP::before (int *a*, int *b*) [private]

This function returns TRUE if a comes before b in their route and FALSE if b is before a. An error is reported if a and b are in different routes. Should be used sparingly as it loops and can be slow for large routes.

Definition at line 3082 of file VRP.cpp.

3.21.3.4 void VRP::capture_best_solution () [private]

Determines if the current solution is the best found so far.

Definition at line 3242 of file VRP.cpp.

3.21.3.5 bool VRP::check_feasibility (VRPViolation * *VV*) [private]

The function returns true if the routes are all feasible and false if any of them are infeasible, placing the worst violations in the VRPViolation VV

Definition at line 653 of file VRP.cpp.

3.21.3.6 bool VRP::check_fixed_edges (const char * *message*)

Makes sure that all fixed edges are still in the solution. If fixed edges are missing from the solution, then some information is displayed and the provided message is printed before exiting.

Definition at line 4051 of file VRP.cpp.

3.21.3.7 bool VRP::check_move (VRPMove * *M*, int *rules*) [private]

Evaluates the move in terms of the rules. Can consider savings, as well as other aspects of the VRPMove M.

Definition at line 1763 of file VRP.cpp.

3.21.3.8 bool VRP::check_savings (VRPMove * *M*, int *rules*) [inline, private]

Evaluates the given savings in terms of the rules. The only part of the rules considered are things such as VRPH_DOWNHILL, VRPH_RECORD_TO_RECORD, VRPH_-SIMULATED_ANNEALING

Definition at line 279 of file VRP.h.

3.21.3.9 bool VRP::check_tabu_status (VRPMove * *M*, int * *old_sol*) [private]

The tabu search rules is entirely route-based. We hash each of the affected routes and see if the values are in the tabu list. If the move is tabu, then we revert back to the old solution and return false. Otherwise, we allow the move and return true. When the move is allowed, we update the tabu list using a circular buffer.

Definition at line 4290 of file VRP.cpp.

3.21.3.10 void VRP::clean_route (int *r*, int *heuristics*)

Runs the provided set of heuristics on route r until a local minimum is reached.

Definition at line 2916 of file VRP.cpp.

3.21.3.11 bool VRP::clone (VRP * *W*)

Copy Constructor for VRP.

Definition at line 317 of file VRP.cpp.

3.21.3.12 void VRP::compute_route_center (int *r*) [private]

Computes the mean x and y coord's of the nodes in a route, storing the information in the VRPRoute[] array

Definition at line 3158 of file VRP.cpp.

3.21.3.13 int VRP::count_num_routes () [private]

Manually counts the # of routes in the current solution.

Definition at line 1452 of file VRP.cpp.

3.21.3.14 bool VRP::create_default_routes (int *day*)

This function creates routes VRPH_DEPOT-i-VRPH_DEPOT for all nodes that require service on the provided day. Returns true if successful, false if the default routes violate some capacity or route length constraint.

Definition at line 1334 of file VRP.cpp.

3.21.3.15 bool VRP::create_default_routes ()

This function creates routes VRPH_DEPOT-i-VRPH_DEPOT for all nodes i and properly initializes all the associated arrays. Returns true if successful, false if the default routes violate some capacity or route length constraint.

Definition at line 1208 of file VRP.cpp.

3.21.3.16 void VRP::create_distance_matrix (int *type*)

Creates the O(n^2) size distance matrix for the provided data using the distance function referenced by type. If the type is EXPLICIT, then the entire distance matrix should be provided in the actual TSPLIB file.

Definition at line 370 of file VRP.cpp.

3.21.3.17 void VRP::create_neighbor_lists (int *nsize*)

Creates the neighbor list of size nsize for each node including the VRPH_DEPOT.

Definition at line 406 of file VRP.cpp.

3.21.3.18 void VRP::create_pred_array ()

This function creates a pred_array from the existing next_array

Definition at line 813 of file VRP.cpp.

3.21.3.19 bool VRP::create_search_neighborhood (int *j*, int *rules*) [private]

Creates the search_size and search_space fields for the current VRP in terms of the given node j and the rules.

Definition at line 2642 of file VRP.cpp.

3.21.3.20 void VRP::eject_neighborhood (int *j*, int *num*, int * *nodelist*)

Ejects num different randomly selected nodes that are in the vicinity of node j, placing the list of ejected nodes in the nodelist[] array. Node j is also ejected! The candidates for ejection are randomly chosen from the 2*num nearest neighbors of j.

Definition at line 2469 of file VRP.cpp.

3.21.3.21 bool VRP::eject_node (int *k*)

This function removes node j from the current solution and adjusts the solution information appropriately.

Definition at line 1627 of file VRP.cpp.

3.21.3.22 bool VRP::eject_route (int *r*, int * *route_buff*)

Ejects all nodes from the current solution that are in route r. Places an ordered list of the ejected nodes in route_buff[].

Definition at line 1732 of file VRP.cpp.

3.21.3.23 double VRP::ejection_cost (int *u*) [private]

Returns the reduction in the objective function value obtained by ejecting u from the current solution.

Definition at line 2896 of file VRP.cpp.

3.21.3.24 void VRP::export_canonical_solution_buff (int * *sol_buff*)

Puts the solution into the buffer in a "canonical form". The orientation of each route is such that start<end. Also, the ordering of the different routes is determined so that route i precedes route j in the ordering if start_i < start_j.

Definition at line 1238 of file VRPIO.cpp.

3.21.3.25 void VRP::export_solution_buff (int * *sol_buff*)

Exports the solution to sol_buff.

Definition at line 1213 of file VRPIO.cpp.

3.21.3.26 void VRP::find_cheapest_insertion (int *j*, int * *edge*, double * *costs*, int *rules*) [private]

Finds the cheapest insertion of node j into the current solution. We store both the best feasible insertion as well as the best overall insertion. The value of edge[0] is the start node of the best feasible edge to be broken and edge[1] is the end node. Similarly, edge[2] and edge[3] represent the start and end of the best overall edge, disregarding feasibility (may be the same as edge[0] and edge[1]). The costs of these insertions are placed in costs[0] and costs[1]. If rules==VRPH_USE_NEIGHBOR_LIST, then only the neighbor list (plus the VRPH_DEPOT to guarantee a singleton route) is searched. If rules==VRPH_NO_NEW_ROUTE, then we do not allow the customer to be added in a new singleton route.

Definition at line 2026 of file VRP.cpp.

3.21.3.27 int VRP::find_common_routes (int * *sol1*, int * *sol2*, int * *route_nums*)

Finds the routes that are shared by the two solutions sol1 and sol2. Places the numbers of these routes (numbers from sol1 which are 1-based) into the route_nums[] buffer and returns the number of shared routes.

Definition at line 4113 of file VRP.cpp.

3.21.3.28 void VRP::find_neighboring_routes () [private]

Finds the nearest set of neighboring routes, placing the corresponding set of route numbers in each route's neighboring_routes[] array.

Definition at line 3180 of file VRP.cpp.

3.21.3.29 void VRP::fix_edge (int *start*, int *end*)

Forces that the provided edge always remains in the solution. The edge may not currently be in the solution, but once it is encountered it will remain in the solution until the edge is unfixed. Note that the fixing of edges is only relevant if you use the heuristics with a VRPH_FIXED_EDGES rules.

Definition at line 3529 of file VRP.cpp.

3.21.3.30 void VRP::fix_string (int * *node_string*, int *k*)

Fixes all edges in the node_string[] array. For example, given the array {1,3,5,4,2}, the edges 1-3, 3-5, 5-4, 4-2 will be fixed. The value of k is the number of nodes in the string.

Definition at line 3597 of file VRP.cpp.

3.21.3.31 double VRP::get_best_known ()

Definition at line 295 of file VRP.cpp.

3.21.3.32 double VRP::get_best_sol_buff (int * *sol_buff*)

Copies the best solution buffer found so far into the sol_buff[] array. Assumes that sol_buff[] is of sufficient size. Returns the total route length of this solution.

Definition at line 247 of file VRP.cpp.

3.21.3.33 double VRP::get_best_total_route_length ()

Returns the total route length of the best solution discovered so far.

Definition at line 259 of file VRP.cpp.

3.21.3.34 double VRP::get_distance_between (int *a*, int *b*) [private]

3.21.3.35 double VRP::get_max_route_length ()

Definition at line 306 of file VRP.cpp.

3.21.3.36 int VRP::get_max_veh_capacity ()

Definition at line 301 of file VRP.cpp.

3.21.3.37 int VRP::get_num_days ()

Returns the number of days in the currently loaded instance.

Definition at line 286 of file VRP.cpp.

3.21.3.38 int VRP::get_num_nodes ()

Returns the number of nodes in the instance.

Definition at line 220 of file VRP.cpp.

3.21.3.39 int VRP::get_num_original_nodes ()

Returns the number of original nodes in the instance.

Definition at line 277 of file VRP.cpp.

3.21.3.40 bool VRP::get_segment_info (int *a*, int *b*, struct VRPSegment * *S*) [private]

Calculates the length, load, and # of customers on the segment of the route between nodes a and b. Assumes that a is before b in the route - this is not checked! Example: a-i-j-b has length: d(a,i)+d(i,j)+d(j,b) load: a + i + j + b #: 4

Definition at line 856 of file VRP.cpp.

3.21.3.41 int VRP::get_string_end (int *a*, int *len*) [private]

Finds the node that is (len-1) hops from a so that the string from a to the end contains len nodes. Returns -1 if not possible to get such a string.

Definition at line 935 of file VRP.cpp.

3.21.3.42 int VRP::get_total_number_of_routes ()

Returns the number of routes in the current solution.

Definition at line 268 of file VRP.cpp.

3.21.3.43 double VRP::get_total_route_length ()

Returns the total route length of the current solution.

Definition at line 229 of file VRP.cpp.

3.21.3.44 double VRP::get_total_service_time ()

Returns the total service time of all customers in the instance.

Definition at line 238 of file VRP.cpp.

3.21.3.45 void VRP::import_solution_buff (int * *sol_buff*)

Imports a solution from buffer produced by something like export_solution_buff(). Can be a partial solution if the first element in sol_buff[] is less than num_original_nodes;

Definition at line 1093 of file VRPIO.cpp.

3.21.3.46 bool VRP::inject_node (int *j*)

Takes the node with index j that is currently NOT in the current solution and adds it to the VRP in the best possible feasible position.

Definition at line 1877 of file VRP.cpp.

3.21.3.47 int VRP::inject_set (int *num*, int * *nodelist*, int *rules*, int *attempts*)

Injects num different nodes in the nodelist[] array into the current solution. If rules=VRPH_RANDOM_SEARCH, then we try attempts different random permutations. If rules=VRPH_REGRET_SEARCH, then we try attempts different permutations and can backtrack by undoing certain moves that we end up regretting.

!! The eventual sol_buff is larger !!!!

Definition at line 2279 of file VRP.cpp.

3.21.3.48 bool VRP::insert_node (int *j*, int *i*, int *k*) [private]

Inserts j in between nodes i and k. Both i and k are assumed to be routed while j is not routed.

Definition at line 1897 of file VRP.cpp.

3.21.3.49 double VRP::insertion_cost (int *u*, int *a*, int *b*) [private]

Returns the increased cost to the route containing a-b of inserting node u in between a and b (assumed to be adjacent!)

Definition at line 2840 of file VRP.cpp.

3.21.3.50 int VRP::intersect_solutions (int * *new_sol*, int ** *routes*, int * *sol1*, int * *sol2*, int *min_routes*)

Takes the two solutions sol1 and sol2 and constructs a smaller instance by ejecting the routes that are in both sol1 and sol2. If the resulting solution contains less than min_routes routes, then we add more random routes from sol1 until the solution has

min_routes. Returns the number of routes ejected from sol1 and places these route buffers in the route_list[][] array which is a 0-based array of routes. Note that sol1 and sol2 are assumed to be full solutions to the VRP instance. Imports the smaller solution new_sol before returning.

Definition at line 3826 of file VRP.cpp.

3.21.3.51 bool VRP::is_feasible (VRPMove * *M*, int *rules*) [private]

Determines whether a proposed move is feasible or not. Could add time window feasibility checks here, etc. The rules is currently not used.

Definition at line 1860 of file VRP.cpp.

3.21.3.52 void VRP::list_fixed_edges (int * *fixed_list*)

Looks through the current solution and places all edges that are currently fixed and in the solution in the fixed_list[] array. So if fixed[2][3]==true and fixed[3][7]==true, the array fixed_list[] is {2,3,3,7}.

Definition at line 3658 of file VRP.cpp.

3.21.3.53 void VRP::normalize_route_numbers ()

Renumbers the routes so that with R total routes, they are numbered 1,2, ..., R instead of all over the place as they typically are after Clarke Wright.

Definition at line 2537 of file VRP.cpp.

3.21.3.54 bool VRP::osman_insert (int *k*, double *alpha*) [private]

Inserts node k into a new location in the solution according to Osman's savings heuristic. Given existing edges i-k-j and l-m, with parameter alpha, we move k to new location l-k-m such that the quantity (ik+kj-lm) - alpha*(lk+km-ij) is minimized.

Definition at line 3903 of file VRP.cpp.

3.21.3.55 int VRP::osman_perturb (int *num*, double *alpha*) [private]

Perturbs the existing solution by attempting to move num different random nodes into new positions using Osman parameter alpha. Gives up after attempting 2*V.num_nodes moves.

Definition at line 4014 of file VRP.cpp.

3.21.3.56 bool VRP::perturb ()

Perturbs the current solution as suggested in Li, et al.

Definition at line 1482 of file VRP.cpp.

3.21.3.57 void VRP::perturb_locations (double *c*) [private]

Perturbs the current problem instance by moving each node slightly as in the TSP paper of Codenotti. The perturbations are scaled in terms of the value c. The distance matrix is recomputed after all locations have been perturbed.

Definition at line 3706 of file VRP.cpp.

3.21.3.58 bool VRP::plot (const char * *filename*)

Assumes default options and plots the .ps file.

Definition at line 280 of file VRPGraphics.cpp.

3.21.3.59 bool VRP::plot (const char * *filename*, int *options*, int *orientation*)

Uses PLPLOT to draw the solution in a .ps file (no other formats supported). Valid options are VRPH_BLACK_AND_WHITE, VRPH_COLOR, VRPH_BARE_BONES, AXES, VRPH_BOXED, TITLED, VRPH_NO_POINTS, VRPH_NO_DEPOT_EDGES, VRPH_WEIGHTED. If options==0, then the default is to draw a VRPH_BOXED, COLORed plot, with AXES and a TITLE. Setting VRPH_BOXED draws a box around the plot with no axes, and setting VRPH_BARE_BONES draws no BOX or AXES. If VRPH_NO_POINTS is set, then the nodes are not drawn on the plot (default is to plot the points). Setting the option VRPH_NO_DEPOT_EDGES will draw each route with the first and last edges of each route not shown and a VRPH_WEIGHTED plot will make the size of the points proportional to their demand. The value of orientation should be 0, 1, 2, or 3, indicating that the image should be rotated through an angle of Pi/2 times orientation. A very primitive attempt is made to scale the line width and point size based on the problem size (number of nodes)

Definition at line 15 of file VRPGraphics.cpp.

3.21.3.60 bool VRP::plot_route (int *r*, const char * *filename*)

Uses PLPLOT to plot the individual route number r in a colored .ps file.

Definition at line 300 of file VRPGraphics.cpp.

3.21.3.61 bool VRP::postsert_dummy (int *i*) [private]

Definition at line 1037 of file VRP.cpp.

3.21.3.62 bool VRP::presert_dummy (int *i*) [private]

Definition at line 1097 of file VRP.cpp.

3.21.3.63 void VRP::print_stats ()

Prints the # of evaluations and moves performed for each heuristic operator.

Definition at line 4368 of file VRP.cpp.

3.21.3.64 int VRP::read_fixed_edges (const char * *filename*)

Reads a file of edges to be fixed. Letting k be the number of edges to be fixed, the file has the format

k start_1 end_1 start_2 end_2 ... start_k end_k

Returns the number of edges that are fixed.

Definition at line 3617 of file VRP.cpp.

3.21.3.65 void VRP::read_solution_file (const char * *filename*)

Imports a solution from filename. File is assumed to be in the format produced by VRP.write_solution_file

Definition at line 1055 of file VRPIO.cpp.

3.21.3.66 void VRP::read_TSPLIB_file (const char * *infile*)

Processes each section of the provided TSPLIB file and records the relevant data in the VRP structure. See the example files for information on my interpretation of the TSPLIB standard as it applies to VRP's.

Definition at line 15 of file VRPIO.cpp.

3.21.3.67 void VRP::refresh_routes ()

Ignores the current route length and load values and recalculates them directly from the next_array.

Definition at line 700 of file VRP.cpp.

3.21.3.68 bool VRP::remove_dummy () [private]

Definition at line 1155 of file VRP.cpp.

3.21.3.69 void VRP::reset ()

Liquidates the solution memory and sets all nodes to unrouted.

Definition at line 4396 of file VRP.cpp.

3.21.3.70 void VRP::reverse_route (int *i*)

This function reverses route number i - does no error checking since we don't know if the routes have normalized numbers or not...

Definition at line 962 of file VRP.cpp.

3.21.3.71 double VRP::RTR_solve (int *heuristics*, int *intensity*, int *max_stuck*, int *num_perturbs*, double *dev*, int *nlist_size*, int *perturb_type*, int *accept_type*, bool *verbose*)

Uses the given parameters to generate a VRP solution via record-to-record travel. Assumes that data has already been imported into V and that we have some existing solution. Returns the objective function value of the best solution found

Definition at line 15 of file VRPSolvers.cpp.

3.21.3.72 double VRP::SA_solve (int *heuristics*, double *start_temp*, double *cool_ratio*, int *iters_per_loop*, int *num_loops*, int *nlist_size*, bool *verbose*)

Uses the given parameters to generate a VRP solution using Simulated Annealing. Assumes that data has already been imported into V and that we have some existing solution. Returns the total route length of the best solution found.

Definition at line 493 of file VRPSolvers.cpp.

3.21.3.73 void VRP::set_best_total_route_length (double *val*)

Definition at line 311 of file VRP.cpp.

3.21.3.74 void VRP::set_daily_demands (int *day*)

Sets the demand of each node equal to the daily demand value for the given day. Used for period VRPs. The day should be a positive integer. If a day of 0 is given, then we set the demand to the mean value.

Definition at line 4180 of file VRP.cpp.

3.21.3.75 void VRP::set_daily_service_times (int *day*)

Sets the service time of each node equal to the daily service time for the given day. Used for period VRPs.

Definition at line 4226 of file VRP.cpp.

3.21.3.76 void VRP::show_next_array ()

Debugging function that shows the current next_array[]

Definition at line 16 of file VRPDebug.cpp.

3.21.3.77 void VRP::show_pred_array ()

Debugging function that shows the pred_array.

Definition at line 32 of file VRPDebug.cpp.

3.21.3.78 void VRP::show_route (int *k*)

Displays information about route number k.

Definition at line 1396 of file VRPIO.cpp.

3.21.3.79 void VRP::show_routes ()

Displays all routes in the solution.

Definition at line 1295 of file VRPIO.cpp.

3.21.3.80 double VRP::split (double *p*)

Splits an existing VRP into two parts by drawing a random ray from the VRPH_-DEPOT. We repeatedly try to split the problem in this way until we have two sets of nodes that each has k nodes where k is between p∗num_nodes and (1-p)∗num_nodes. The value of p must be less than .5. Returns the angle theta used to split the problem

Definition at line 3371 of file VRP.cpp.

3.21.3.81 int VRP::split_routes (double *p*, int ** *ejected_routes*, double * *t*)

Splits an existing VRP into two parts by drawing a random ray from the VRPH_-DEPOT. We repeatedly try to split the problem in this way until we have two sets of nodes that each has k nodes where k is between p*num_nodes and (1-p)*num_nodes. Next, we take the larger part of the split solution and then keep all the routes that have at least one node in the larger part. The route-based decisions are based on the currently loaded solution. The ejected routes are placed in the ejected_routes[][] array and the function returns the number of routes ejected. The final value of theta that splits the problem is placed in t.

Definition at line 3436 of file VRP.cpp.

3.21.3.82 void VRP::summary ()

This function prints out a summary of the current solution and the individual routes.

Definition at line 1429 of file VRPIO.cpp.

3.21.3.83 void VRP::unfix_all ()

Unfixes any and all edges that may be currently fixed.

Definition at line 3586 of file VRP.cpp.

3.21.3.84 void VRP::unfix_edge (int *start*, int *end*)

Unfixes an edge that is already fixed.

Definition at line 3558 of file VRP.cpp.

3.21.3.85 void VRP::update (VRPMove * *M*) [private]

Updates the solution information in terms of the move M.

Definition at line 3122 of file VRP.cpp.

3.21.3.86 void VRP::update_arrival_times () [private]

Computes the arrival time at all customers.

Definition at line 4246 of file VRP.cpp.

3.21.3.87 void VRP::update_route (int *j*, VRPRoute * *R*)

Copies the fields of route j in the current solution to the VRPRoute R, also updating the ordering, x, and y arrays. The ordering is normalized by having start<end.

Definition at line 3301 of file VRP.cpp.

3.21.3.88 void VRP::update_solution_wh () [private]

Attempts to add the given solution to the warehouse.

Definition at line 3277 of file VRP.cpp.

3.21.3.89 bool VRP::verify_routes (const char * *message*)

This debugging function will manually calculate the objective function of the current solution and the route values, etc., and compare with the claimed value. Returns false if any inconsistencies are found and prints the message. Returns true with no output otherwise.

Definition at line 50 of file VRPDebug.cpp.

3.21.3.90 void VRP::write_solution_file (const char * *filename*)

Exports a solution to the designated filename in canonical form. Let N be the # of non-VRPH_DEPOT nodes in the problem. Then the first entry in the file is N and the following N+1 entries simply traverse the solution in order where we enter a node's negative index if it is the first node in a route. The solution is put into canonical form - the routes are traversed in the orientation where the start index is less than the end index, and the routes are sorted by the start index. Example: Route 1: 0-3-2-0, Route 2: 0-4-1-0 File is then: 4 -1 4 -2 3 0

Definition at line 849 of file VRPIO.cpp.

3.21.3.91 void VRP::write_solutions (int *num_sols*, const char * *filename*)

Writes num_sols solutions from the solution warehouse to the designated filename. The format is the same as for write_solution_file.

Definition at line 920 of file VRPIO.cpp.

3.21.3.92 void VRP::write_tex_file (const char * *filename*)

Writes the solution in a TeX tabular format using the longtable package in case the solution spans multiple pages.

Definition at line 983 of file VRPIO.cpp.

3.21.3.93 void VRP::write_TSPLIB_file (const char * *outfile*)

Exports the data from an already loaded instance to a CVRP outfile in TSPLIB format (using EXACT_2D distance).

Definition at line 793 of file VRPIO.cpp.

3.21.4 Friends And Related Function Documentation

3.21.4.1 friend class ClarkeWright [friend]

Definition at line 36 of file VRP.h.

3.21.4.2 friend class Concatenate [friend]

Definition at line 33 of file VRP.h.

3.21.4.3 friend class CrossExchange [friend]

Definition at line 26 of file VRP.h.

3.21.4.4 friend class Flip [friend]

Definition at line 34 of file VRP.h.

3.21.4.5 friend class MoveString [friend]

Definition at line 30 of file VRP.h.

3.21.4.6 friend class OnePointMove [friend]

Definition at line 20 of file VRP.h.

3.21.4.7 friend class OrOpt [friend]

Definition at line 25 of file VRP.h.

3.21.4.8 friend class Postsert [friend]

Definition at line 28 of file VRP.h.

3.21.4.9 friend class Presert [friend]

Definition at line 29 of file VRP.h.

3.21.4.10 friend class Swap [friend]

Definition at line 31 of file VRP.h.

3.21.4.11 friend class SwapEnds [friend]

Definition at line 32 of file VRP.h.

3.21.4.12 friend class Sweep [friend]

Definition at line 37 of file VRP.h.

3.21.4.13 friend class ThreeOpt [friend]

Definition at line 24 of file VRP.h.

3.21.4.14 friend class ThreePointMove [friend]

Definition at line 22 of file VRP.h.

3.21.4.15 friend class TwoOpt [friend]

Definition at line 23 of file VRP.h.

3.21.4.16 friend class TwoPointMove [friend]

Definition at line 21 of file VRP.h.

3.21.5 Member Data Documentation

3.21.5.1 double VRP::balance_parameter [private]

Definition at line 193 of file VRP.h.

3.21.5.2 double VRP::best_known [private]

Definition at line 176 of file VRP.h.

3.21.5.3 int* VRP::best_sol_buff [private]

Definition at line 172 of file VRP.h.

3.21.5.4 double VRP::best_total_route_length [private]

Definition at line 173 of file VRP.h.

3.21.5.5 bool VRP::can_display [private]

Definition at line 202 of file VRP.h.

3.21.5.6 double VRP::cooling_ratio [private]

Definition at line 197 of file VRP.h.

3.21.5.7 int VRP::coord_type [private]

Definition at line 189 of file VRP.h.

3.21.5.8 int* VRP::current_sol_buff [private]

Definition at line 239 of file VRP.h.

3.21.5.9 double** VRP::d [private]

Definition at line 204 of file VRP.h.

3.21.5.10 bool VRP::depot_normalized [private]

Definition at line 210 of file VRP.h.

3.21.5.11 double VRP::deviation [private]

Definition at line 234 of file VRP.h.

3.21.5.12 int VRP::display_type [private]

Definition at line 190 of file VRP.h.

3.21.5.13 int VRP::dummy_index [private]

Definition at line 194 of file VRP.h.

3.21.5.14 int VRP::edge_weight_format [private]

Definition at line 191 of file VRP.h.

3.21.5.15 int VRP::edge_weight_type [private]

Definition at line 188 of file VRP.h.

3.21.5.16 bool∗∗ VRP::fixed [private]

Definition at line 205 of file VRP.h.

3.21.5.17 double VRP::fixed_service_time [private]

Definition at line 187 of file VRP.h.

3.21.5.18 bool VRP::forbid_tiny_moves [private]

Definition at line 213 of file VRP.h.

3.21.5.19 bool VRP::has_service_times [private]

Definition at line 186 of file VRP.h.

3.21.5.20 int VRP::matrix_size [private]

Definition at line 192 of file VRP.h.

3.21.5.21 double VRP::max_route_length [private]

Definition at line 182 of file VRP.h.

3.21.5.22 double VRP::max_theta [private]

Definition at line 237 of file VRP.h.

3.21.5.23 int VRP::max_veh_capacity [private]

Definition at line 180 of file VRP.h.

3.21.5.24 double VRP::min_route_length [private]

Definition at line 183 of file VRP.h.

3.21.5.25 double VRP::min_theta [private]

Definition at line 236 of file VRP.h.

3.21.5.26 int VRP::min_vehicles [private]

Definition at line 185 of file VRP.h.

3.21.5.27 char VRP::name[VRPH_STRING_SIZE]

Definition at line 97 of file VRP.h.

3.21.5.28 int VRP::neighbor_list_size [private]

Definition at line 195 of file VRP.h.

3.21.5.29 int* VRP::next_array [private]

Definition at line 222 of file VRP.h.

3.21.5.30 class VRPNode* VRP::nodes [private]

Definition at line 207 of file VRP.h.

3.21.5.31 int VRP::num_days [private]

Definition at line 177 of file VRP.h.

3.21.5.32 int VRP::num_evaluations[NUM_HEURISTICS]

Definition at line 163 of file VRP.h.

3.21.5.33 int VRP::num_moves[NUM_HEURISTICS]

Definition at line 164 of file VRP.h.

3.21.5.34 int VRP::num_nodes [private]

Definition at line 169 of file VRP.h.

3.21.5.35 int VRP::num_original_nodes [private]

Definition at line 175 of file VRP.h.

3.21.5.36 double VRP::orig_max_route_length [private]

Definition at line 184 of file VRP.h.

3.21.5.37 int VRP::orig_max_veh_capacity [private]

Definition at line 181 of file VRP.h.

3.21.5.38 int* VRP::pred_array [private]

Definition at line 223 of file VRP.h.

3.21.5.39 int VRP::problem_type [private]

Definition at line 178 of file VRP.h.

3.21.5.40 double VRP::record [private]

Definition at line 233 of file VRP.h.

3.21.5.41 class VRPRoute* VRP::route [private]

Definition at line 227 of file VRP.h.

3.21.5.42 int* VRP::route_num [private]

Definition at line 224 of file VRP.h.

3.21.5.43 VRPRouteWarehouse* VRP::route_wh

Definition at line 147 of file VRP.h.

3.21.5.44 bool* VRP::routed [private]

Definition at line 225 of file VRP.h.

3.21.5.45 int VRP::search_size [private]

Definition at line 218 of file VRP.h.

3.21.5.46 int* VRP::search_space [private]

Definition at line 219 of file VRP.h.

3.21.5.47 VRPSolutionWarehouse* VRP::solution_wh

Definition at line 146 of file VRP.h.

3.21.5.48 bool VRP::symmetric [private]

Definition at line 199 of file VRP.h.

3.21.5.49 class VRPTabuList* VRP::tabu_list [private]

Definition at line 230 of file VRP.h.

3.21.5.50 double VRP::temperature [private]

Definition at line 196 of file VRP.h.

3.21.5.51 int VRP::total_demand [private]

Definition at line 179 of file VRP.h.

3.21.5.52 int VRP::total_number_of_routes [private]

Definition at line 174 of file VRP.h.

3.21.5.53 double VRP::total_route_length [private]

Definition at line 170 of file VRP.h.

3.21.5.54 double VRP::total_service_time [private]

Definition at line 171 of file VRP.h.

3.21.5.55 class VRPViolation VRP::violation [private]

Definition at line 246 of file VRP.h.

The documentation for this class was generated from the following files:

- inc/VRP.h
- src/VRP.cpp
- src/VRPDebug.cpp
- src/VRPGraphics.cpp
- src/VRPIO.cpp
- src/VRPSolvers.cpp

3.22 VRPMove Class Reference

`#include <VRPMove.h>`

Public Member Functions

- VRPMove ()
- VRPMove (int n)
- ~VRPMove ()
- bool is_better (class VRP *V, VRPMove *M2, int criteria)

Public Attributes

- int criteria
- int num_affected_routes
- int route_nums [MAX_AFFECTED_ROUTES]
- double route_lens [MAX_AFFECTED_ROUTES]
- int route_loads [MAX_AFFECTED_ROUTES]
- int route_custs [MAX_AFFECTED_ROUTES]
- double * arrival_times
- double savings
- int total_number_of_routes
- double new_total_route_length
- int move_type
- int num_arguments
- int move_arguments [MAX_ARGUMENTS]
- int eval_arguments [MAX_ARGUMENTS]
- bool evaluated_savings

3.22.1 Detailed Description

Definition at line 18 of file VRPMove.h.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 VRPMove::VRPMove ()

Contains data regarding a particular "move" or solution modification.

Definition at line 15 of file VRPMove.cpp.

3.22.2.2 VRPMove::VRPMove (int *n*)

Definition at line 29 of file VRPMove.cpp.

3.22.2.3 VRPMove::~VRPMove ()

Definition at line 42 of file VRPMove.cpp.

3.22.3 Member Function Documentation

3.22.3.1 bool VRPMove::is_better (class VRP * *V*, VRPMove * *M2*, int *criteria*)

Evaluates this move versus M2 in terms of the provided rules. Returns true of this move is superior to M2 and false otherwise.

Definition at line 49 of file VRPMove.cpp.

3.22.4 Member Data Documentation

3.22.4.1 double* VRPMove::arrival_times

Definition at line 40 of file VRPMove.h.

3.22.4.2 int VRPMove::criteria

Definition at line 33 of file VRPMove.h.

3.22.4.3 int VRPMove::eval_arguments[MAX_ARGUMENTS]

Definition at line 48 of file VRPMove.h.

3.22.4.4 bool VRPMove::evaluated_savings

Definition at line 50 of file VRPMove.h.

3.22.4.5 int VRPMove::move_arguments[MAX_ARGUMENTS]

Definition at line 47 of file VRPMove.h.

3.22.4.6 int VRPMove::move_type

Definition at line 45 of file VRPMove.h.

3.22.4.7 double VRPMove::new_total_route_length

Definition at line 44 of file VRPMove.h.

3.22.4.8 int VRPMove::num_affected_routes

Definition at line 34 of file VRPMove.h.

3.22.4.9 int VRPMove::num_arguments

Definition at line 46 of file VRPMove.h.

3.22.4.10 int VRPMove::route_custs[MAX_AFFECTED_ROUTES]

Definition at line 38 of file VRPMove.h.

3.22.4.11 double VRPMove::route_lens[MAX_AFFECTED_ROUTES]

Definition at line 36 of file VRPMove.h.

3.22.4.12 int VRPMove::route_loads[MAX_AFFECTED_ROUTES]

Definition at line 37 of file VRPMove.h.

3.22.4.13 int VRPMove::route_nums[MAX_AFFECTED_ROUTES]

Definition at line 35 of file VRPMove.h.

3.22.4.14 double VRPMove::savings

Definition at line 42 of file VRPMove.h.

3.22.4.15 int VRPMove::total_number_of_routes

Definition at line 43 of file VRPMove.h.

The documentation for this class was generated from the following files:

- inc/VRPMove.h
- src/VRPMove.cpp

3.23 VRPNeighborElement Class Reference

`#include <VRPUtils.h>`

Public Attributes

- double val
- int position

3.23.1 Detailed Description

Definition at line 70 of file VRPUtils.h.

3.23.2 Member Data Documentation

3.23.2.1 int VRPNeighborElement::position

Definition at line 74 of file VRPUtils.h.

3.23.2.2 double VRPNeighborElement::val

Definition at line 73 of file VRPUtils.h.

The documentation for this class was generated from the following file:

- inc/VRPUtils.h

3.24 VRPNeighborhood Class Reference

`#include <VRPUtils.h>`

Public Member Functions

- VRPNeighborhood (int n)

Public Attributes

- int move_type
- int node_1
- int node_2
- class VRPMove * Moves
- int size

3.24.1 Detailed Description

Definition at line 94 of file VRPUtils.h.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 VRPNeighborhood::VRPNeighborhood (int *n*)

3.24.3 Member Data Documentation

3.24.3.1 int VRPNeighborhood::move_type

Definition at line 97 of file VRPUtils.h.

3.24.3.2 class VRPMove* VRPNeighborhood::Moves

Definition at line 99 of file VRPUtils.h.

3.24.3.3 int VRPNeighborhood::node_1

Definition at line 98 of file VRPUtils.h.

3.24.3.4 int VRPNeighborhood::node_2

Definition at line 98 of file VRPUtils.h.

3.24.3.5 int VRPNeighborhood::size

Definition at line 100 of file VRPUtils.h.

The documentation for this class was generated from the following file:

- inc/VRPUtils.h

3.25 VRPNode Class Reference

`#include <VRPNode.h>`

Public Member Functions

- VRPNode ()
- VRPNode (int d)
- ~VRPNode ()
- void duplicate (VRPNode *N)
- void show ()

Public Attributes

- double x
- double y
- double r
- double theta
- int id
- int demand
- int * daily_demands
- int cluster
- VRPNeighborElement neighbor_list [MAX_NEIGHBORLIST_SIZE]
- double service_time
- double * daily_service_times
- double arrival_time
- double start_tw
- double end_tw
- int num_days

3.25.1 Detailed Description

Definition at line 19 of file VRPNode.h.

3.25.2 Constructor & Destructor Documentation

3.25.2.1 VRPNode::VRPNode ()

Default constructor for the VRPNode class. Allocates an array of MAX_-NEIGHBORLIST_SIZE VRPNeighborElements for the node.

Definition at line 15 of file VRPNode.cpp.

3.25.2.2 VRPNode::VRPNode (int *d*)

3.25.2.3 VRPNode::~VRPNode ()

VRPNode destructor.

Definition at line 43 of file VRPNode.cpp.

3.25.3 Member Function Documentation

3.25.3.1 void VRPNode::duplicate (VRPNode * *N*)

3.25.3.2 void VRPNode::show ()

3.25.4 Member Data Documentation

3.25.4.1 double VRPNode::arrival_time

Definition at line 37 of file VRPNode.h.

3.25.4.2 int VRPNode::cluster

Definition at line 30 of file VRPNode.h.

3.25.4.3 int* VRPNode::daily_demands

Definition at line 29 of file VRPNode.h.

3.25.4.4 double* VRPNode::daily_service_times

Definition at line 35 of file VRPNode.h.

3.25.4.5 int VRPNode::demand

Definition at line 28 of file VRPNode.h.

3.25.4.6 double VRPNode::end_tw

Definition at line 39 of file VRPNode.h.

3.25.4.7 int VRPNode::id

Definition at line 27 of file VRPNode.h.

3.25.4.8 VRPNeighborElement VRPNode::neighbor_list[MAX_-NEIGHBORLIST_SIZE]

Definition at line 31 of file VRPNode.h.

3.25.4.9 int VRPNode::num_days

Definition at line 41 of file VRPNode.h.

3.25.4.10 double VRPNode::r

Definition at line 25 of file VRPNode.h.

3.25.4.11 double VRPNode::service_time

Definition at line 33 of file VRPNode.h.

3.25.4.12 double VRPNode::start_tw

Definition at line 38 of file VRPNode.h.

3.25.4.13 double VRPNode::theta

Definition at line 26 of file VRPNode.h.

3.25.4.14 double VRPNode::x

Definition at line 23 of file VRPNode.h.

3.25.4.15 double VRPNode::y

Definition at line 24 of file VRPNode.h.

The documentation for this class was generated from the following files:

- inc/VRPNode.h

- src/VRPNode.cpp

3.26 VRPRoute Class Reference

`#include <VRPRoute.h>`

Public Member Functions

- VRPRoute ()
- VRPRoute (int n)
- ~VRPRoute ()
- void create_name ()
- int hash (int salt)

Public Attributes

- int start
- int end
- double length
- int load
- int num_customers
- double obj_val
- int hash_val
- int hash_val2
- double total_service_time
- double time
- double * x
- double * y
- char * name
- double x_center
- double y_center
- double min_theta
- double max_theta
- int neighboring_routes [MAX_NEIGHBORING_ROUTES]
- int * ordering

3.26.1 Detailed Description

Definition at line 24 of file VRPRoute.h.

3.26.2 Constructor & Destructor Documentation

3.26.2.1 VRPRoute::VRPRoute ()

Stores information about a particular route. The ordering field is not updated during the search and is filled in only when requested.

Default constructor for the VRPRoute.

Definition at line 17 of file VRPRoute.cpp.

3.26.2.2 VRPRoute::VRPRoute (int *n*)

Allocates memory for the VRPRoute fields large enough for an n node problem.

Definition at line 30 of file VRPRoute.cpp.

3.26.2.3 VRPRoute::~VRPRoute ()

Destructor for the VRPRoute.

Definition at line 45 of file VRPRoute.cpp.

3.26.3 Member Function Documentation

3.26.3.1 void VRPRoute::create_name ()

Creates a name for the route as a string. Format is hashval1_hashval2_ordering (delimited by _)

Definition at line 108 of file VRPRoute.cpp.

3.26.3.2 int VRPRoute::hash (int *salt*)

Computes a hash of the route, returning an integer in the range [0,HASH_TABLE_SIZE-1].

Definition at line 63 of file VRPRoute.cpp.

3.26.4 Member Data Documentation

3.26.4.1 int VRPRoute::end

Definition at line 38 of file VRPRoute.h.

3.26.4.2 int VRPRoute::hash_val

Definition at line 44 of file VRPRoute.h.

3.26.4.3 int VRPRoute::hash_val2

Definition at line 45 of file VRPRoute.h.

3.26.4.4 double VRPRoute::length

Definition at line 39 of file VRPRoute.h.

3.26.4.5 int VRPRoute::load

Definition at line 40 of file VRPRoute.h.

3.26.4.6 double VRPRoute::max_theta

Definition at line 58 of file VRPRoute.h.

3.26.4.7 double VRPRoute::min_theta

Definition at line 57 of file VRPRoute.h.

3.26.4.8 char* VRPRoute::name

Definition at line 52 of file VRPRoute.h.

3.26.4.9 int VRPRoute::neighboring_routes[MAX_NEIGHBORING_ROUTES]

Definition at line 60 of file VRPRoute.h.

3.26.4.10 int VRPRoute::num_customers

Definition at line 41 of file VRPRoute.h.

3.26.4.11 double VRPRoute::obj_val

Definition at line 42 of file VRPRoute.h.

3.26.4.12 int* VRPRoute::ordering

Definition at line 62 of file VRPRoute.h.

3.26.4.13 int VRPRoute::start

Definition at line 37 of file VRPRoute.h.

3.26.4.14 double VRPRoute::time

Definition at line 48 of file VRPRoute.h.

3.26.4.15 double VRPRoute::total_service_time

Definition at line 47 of file VRPRoute.h.

3.26.4.16 double* VRPRoute::x

Definition at line 49 of file VRPRoute.h.

3.26.4.17 double VRPRoute::x_center

Definition at line 54 of file VRPRoute.h.

3.26.4.18 double* VRPRoute::y

Definition at line 50 of file VRPRoute.h.

3.26.4.19 double VRPRoute::y_center

Definition at line 55 of file VRPRoute.h.

The documentation for this class was generated from the following files:

- inc/VRPRoute.h
- src/VRPRoute.cpp

3.27 VRPRouteWarehouse Class Reference

`#include <VRPRoute.h>`

Public Member Functions

- VRPRouteWarehouse ()
- VRPRouteWarehouse (int h_size)
- ~VRPRouteWarehouse ()
- void remove_route (int hash_val, int hash_val2)
- int add_route (VRPRoute *R)
- void liquidate ()

Public Attributes

- int hash_table_size
- int num_unique_routes
- struct htable_entry * hash_table

3.27.1 Detailed Description

Definition at line 71 of file VRPRoute.h.

3.27.2 Constructor & Destructor Documentation

3.27.2.1 VRPRouteWarehouse::VRPRouteWarehouse ()

Default constructor for the route warehouse.

Definition at line 141 of file VRPRoute.cpp.

3.27.2.2 VRPRouteWarehouse::VRPRouteWarehouse (int *h_size*)

Constructor for the rote warehouse with h_size entries in the hash table. Best if h_size is a power of 2.

Definition at line 152 of file VRPRoute.cpp.

3.27.2.3 VRPRouteWarehouse::~VRPRouteWarehouse ()

Destructor for the route warehouse.

Definition at line 171 of file VRPRoute.cpp.

3.27.3 Member Function Documentation

3.27.3.1 int VRPRouteWarehouse::add_route (VRPRoute * *R*)

Adds the given route R to the warehouse, returning true if the addition is made and false otherwise.

Definition at line 248 of file VRPRoute.cpp.

3.27.3.2 void VRPRouteWarehouse::liquidate ()

Clears the hash table, removing all routes from the WH.

Definition at line 181 of file VRPRoute.cpp.

3.27.3.3 void VRPRouteWarehouse::remove_route (int *hash_val*, int *hash_val2*)

Removes a particular route from the hash table, using the second hash value to remove the correct entry if we have duplicates at the same location in the table.

Definition at line 194 of file VRPRoute.cpp.

3.27.4 Member Data Documentation

3.27.4.1 struct htable_entry* VRPRouteWarehouse::hash_table [read]

Definition at line 80 of file VRPRoute.h.

3.27.4.2 int VRPRouteWarehouse::hash_table_size

Definition at line 78 of file VRPRoute.h.

3.27.4.3 int VRPRouteWarehouse::num_unique_routes

Definition at line 79 of file VRPRoute.h.

The documentation for this class was generated from the following files:

- inc/VRPRoute.h
- src/VRPRoute.cpp

3.28 VRPSavingsElement Class Reference

`#include <VRPUtils.h>`

Public Attributes

- double savings
- int position
- int i
- int j

3.28.1 Detailed Description

Definition at line 58 of file VRPUtils.h.

3.28.2 Member Data Documentation

3.28.2.1 int VRPSavingsElement::i

Definition at line 65 of file VRPUtils.h.

3.28.2.2 int VRPSavingsElement::j

Definition at line 66 of file VRPUtils.h.

3.28.2.3 int VRPSavingsElement::position

Definition at line 64 of file VRPUtils.h.

3.28.2.4 double VRPSavingsElement::savings

Definition at line 63 of file VRPUtils.h.

The documentation for this class was generated from the following file:

- inc/VRPUtils.h

3.29 VRPSeedElement Class Reference

`#include <VRPUtils.h>`

Private Attributes

- double val
- int position
- int demand

3.29.1 Detailed Description

Definition at line 87 of file VRPUtils.h.

3.29.2 Member Data Documentation

3.29.2.1 int VRPSeedElement::demand [private]

Definition at line 91 of file VRPUtils.h.

3.29.2.2 int VRPSeedElement::position [private]

Definition at line 90 of file VRPUtils.h.

3.29.2.3 double VRPSeedElement::val [private]

Definition at line 89 of file VRPUtils.h.

The documentation for this class was generated from the following file:

- inc/VRPUtils.h

3.30 VRPSegment Struct Reference

`#include <VRPUtils.h>`

Public Attributes

- int segment_start
- int segment_end
- int num_custs
- int load
- double len

3.30.1 Detailed Description

Definition at line 105 of file VRPUtils.h.

3.30.2 Member Data Documentation

3.30.2.1 double VRPSegment::len

Definition at line 117 of file VRPUtils.h.

3.30.2.2 int VRPSegment::load

Definition at line 116 of file VRPUtils.h.

3.30.2.3 int VRPSegment::num_custs

Definition at line 115 of file VRPUtils.h.

3.30.2.4 int VRPSegment::segment_end

Definition at line 113 of file VRPUtils.h.

3.30.2.5 int VRPSegment::segment_start

Contains information about a particular segment of a route.

Definition at line 112 of file VRPUtils.h.

The documentation for this struct was generated from the following file:

- inc/VRPUtils.h

3.31 VRPSolution Class Reference

`#include <VRPSolution.h>`

Public Member Functions

- VRPSolution ()
- VRPSolution (int n)
- ~VRPSolution ()
- int hash (int salt)

Public Attributes

- bool in_IP
- double obj
- int n
- int * sol
- double time

3.31.1 Detailed Description

Definition at line 16 of file VRPSolution.h.

3.31.2 Constructor & Destructor Documentation

3.31.2.1 VRPSolution::VRPSolution ()

Contains fields and methods to process solutions to the VRP.

Default constructor for the VRPSolution.

Definition at line 293 of file VRPSolution.cpp.

3.31.2.2 VRPSolution::VRPSolution (int *n*)

Constructor for an n-node VRPSolution.

Definition at line 277 of file VRPSolution.cpp.

3.31.2.3 VRPSolution::~VRPSolution ()

Destructor for the VRPSolution.

Definition at line 307 of file VRPSolution.cpp.

3.31.3 Member Function Documentation

3.31.3.1 int VRPSolution::hash (int *salt*)

Computes a hash of the solution, returning an integer in the range [0,n-1]. The solution buffer should be in canonical form so that the hash value is in terms of the ordering.

Definition at line 255 of file VRPSolution.cpp.

3.31.4 Member Data Documentation

3.31.4.1 bool VRPSolution::in_IP

Definition at line 27 of file VRPSolution.h.

3.31.4.2 int VRPSolution::n

Definition at line 29 of file VRPSolution.h.

3.31.4.3 double VRPSolution::obj

Definition at line 28 of file VRPSolution.h.

3.31.4.4 int* VRPSolution::sol

Definition at line 30 of file VRPSolution.h.

3.31.4.5 double VRPSolution::time

Definition at line 31 of file VRPSolution.h.

The documentation for this class was generated from the following files:

- inc/VRPSolution.h
- src/VRPSolution.cpp

3.32 VRPSolutionWarehouse Class Reference

`#include <VRPSolution.h>`

Public Member Functions

- VRPSolutionWarehouse ()
- ~VRPSolutionWarehouse ()
- VRPSolutionWarehouse (int num_sols, int n)
- int add_sol (VRPSolution *new_sol, int start_index)
- bool liquidate ()
- void sort_sols ()
- void show ()

Public Attributes

- int num_sols
- int max_size
- double worst_obj
- VRPSolution * sols
- struct htable_entry * hash_table

3.32.1 Detailed Description

Definition at line 36 of file VRPSolution.h.

3.32.2 Constructor & Destructor Documentation

3.32.2.1 VRPSolutionWarehouse::VRPSolutionWarehouse ()

Default constructor for the solution warehouse.

Definition at line 16 of file VRPSolution.cpp.

3.32.2.2 VRPSolutionWarehouse::~VRPSolutionWarehouse ()

Destructor for the solution warehouse.

Definition at line 53 of file VRPSolution.cpp.

3.32.2.3 VRPSolutionWarehouse::VRPSolutionWarehouse (int *num_sols*, int *n*)

Constructs a warehouse of max_sols solutions, with sufficient memory for an n-node problem.

Definition at line 29 of file VRPSolution.cpp.

3.32.3 Member Function Documentation

3.32.3.1 int VRPSolutionWarehouse::add_sol (VRPSolution * *new_sol*, int *start_index*)

Attempts to add a solution to the warehouse. Returns the index that the new solution was placed at. Returns -1 if the solution was not placed in the warehouse. The start_index provides a place to begin the search -- useful when inserting multiple solutions whose order is already known. Use start_index=0 if no information about the solution's position is known. The VRPSolution being passed in should be in "canonical form" for the hash function to work properly!!

Definition at line 70 of file VRPSolution.cpp.

3.32.3.2 bool VRPSolutionWarehouse::liquidate ()

Removes all solutions from the warehouse.

Definition at line 211 of file VRPSolution.cpp.

3.32.3.3 void VRPSolutionWarehouse::show ()

Debugging function to show the current solutions in the warehouse.

Definition at line 189 of file VRPSolution.cpp.

3.32.3.4 void VRPSolutionWarehouse::sort_sols ()

Sorts the solutions in the warehouse in increasing order of the objective function value.

Definition at line 244 of file VRPSolution.cpp.

3.32.4 Member Data Documentation

3.32.4.1 struct htable_entry* VRPSolutionWarehouse::hash_table [read]

Definition at line 50 of file VRPSolution.h.

3.32.4.2 int VRPSolutionWarehouse::max_size

Definition at line 47 of file VRPSolution.h.

3.32.4.3 int VRPSolutionWarehouse::num_sols

Definition at line 46 of file VRPSolution.h.

3.32.4.4 VRPSolution* VRPSolutionWarehouse::sols

Definition at line 49 of file VRPSolution.h.

3.32.4.5 double VRPSolutionWarehouse::worst_obj

Definition at line 48 of file VRPSolution.h.

The documentation for this class was generated from the following files:

- inc/VRPSolution.h
- src/VRPSolution.cpp

3.33 VRPTabuList Class Reference

`#include <VRPTabuList.h>`

Public Member Functions

- VRPTabuList ()
- VRPTabuList (int t)
- ~VRPTabuList ()
- void update_list (VRPRoute *r)
- void show ()
- void empty ()

Public Attributes

- int max_entries
- int num_entries
- int start_index
- int * hash_vals1
- int * hash_vals2
- bool full

3.33.1 Detailed Description

Definition at line 18 of file VRPTabuList.h.

3.33.2 Constructor & Destructor Documentation

3.33.2.1 VRPTabuList::VRPTabuList ()

Default constructor for the VRPTabuList.

Definition at line 15 of file VRPTabuList.cpp.

3.33.2.2 VRPTabuList::VRPTabuList (int *t*)

Constructor for the VRPTabuList with t tabu routes.

Definition at line 31 of file VRPTabuList.cpp.

3.33.2.3 VRPTabuList::~VRPTabuList ()

Destructor for the VRPTabuList.

Definition at line 54 of file VRPTabuList.cpp.

3.33.3 Member Function Documentation

3.33.3.1 void VRPTabuList::empty ()

Removes all entries from the tabu list.

Definition at line 113 of file VRPTabuList.cpp.

3.33.3.2 void VRPTabuList::show ()

Shows the hash values of the current tabu list, starting with start_index and listing all current entries.

Definition at line 137 of file VRPTabuList.cpp.

3.33.3.3 void VRPTabuList::update_list (VRPRoute * *r*)

Updates the tabu list by adding the route r.

Definition at line 67 of file VRPTabuList.cpp.

3.33.4 Member Data Documentation

3.33.4.1 bool VRPTabuList::full

Definition at line 34 of file VRPTabuList.h.

3.33.4.2 int* VRPTabuList::hash_vals1

Definition at line 31 of file VRPTabuList.h.

3.33.4.3 int* VRPTabuList::hash_vals2

Definition at line 32 of file VRPTabuList.h.

3.33.4.4 int VRPTabuList::max_entries

Definition at line 28 of file VRPTabuList.h.

3.33.4.5 int VRPTabuList::num_entries

Definition at line 29 of file VRPTabuList.h.

3.33.4.6 int VRPTabuList::start_index

Definition at line 30 of file VRPTabuList.h.

The documentation for this class was generated from the following files:

- inc/VRPTabuList.h
- src/VRPTabuList.cpp

3.34 VRPViolation Class Reference

`#include <VRPUtils.h>`

Public Attributes

- double length_violation
- int capacity_violation

3.34.1 Detailed Description

Definition at line 78 of file VRPUtils.h.

3.34.2 Member Data Documentation

3.34.2.1 int VRPViolation::capacity_violation

Definition at line 82 of file VRPUtils.h.

3.34.2.2 double VRPViolation::length_violation

Definition at line 81 of file VRPUtils.h.

The documentation for this class was generated from the following file:

- inc/VRPUtils.h

Chapter 4

File Documentation

4.1 inc/ClarkeWright.h File Reference

Classes

- class ClarkeWright

Defines

- #define VRPH_UNUSED 0
- #define VRPH_ADDED 1
- #define VRPH_INTERIOR 2

4.1.1 Define Documentation

4.1.1.1 #define VRPH_ADDED 1

Definition at line 18 of file ClarkeWright.h.

4.1.1.2 #define VRPH_INTERIOR 2

Definition at line 19 of file ClarkeWright.h.

4.1.1.3 #define VRPH_UNUSED 0

Definition at line 17 of file ClarkeWright.h.

4.2 inc/Concatenate.h File Reference

Classes

- class Concatenate

4.3 inc/CrossExchange.h File Reference

Classes

- class CrossExchange

4.4 inc/Flip.h File Reference

Classes

- class Flip

4.5 inc/MoveString.h File Reference

Classes

- class MoveString

4.6 inc/OnePointMove.h File Reference

Classes

- class OnePointMove

4.7 inc/OrOpt.h File Reference

Classes

- class OrOpt

4.8 inc/Osman.h File Reference

Functions

- bool osman_insert (VRP *V, int k, double alpha)
- int osman_perturb (VRP *V, int num, double alpha)

4.8.1 Function Documentation

4.8.1.1 bool osman_insert (VRP * *V*, int *k*, double *alpha*)

4.8.1.2 int osman_perturb (VRP * *V*, int *num*, double *alpha*)

4.9 inc/Postsert.h File Reference

Classes

- class Postsert

4.10 inc/Presert.h File Reference

Classes

- class Presert

4.11 inc/RNG.h File Reference

Defines

- #define NUM_RANDVALS 2000

Functions

- double lcgrand (int stream)
- void random_permutation (int *perm, int n)

4.11.1 Define Documentation

4.11.1.1 #define NUM_RANDVALS 2000

Definition at line 16 of file RNG.h.

4.11.2 Function Documentation

4.11.2.1 double lcgrand (int *stream*)

A run of the mill LCG.

Definition at line 79 of file RNG.cpp.

4.11.2.2 void random_permutation (int * *perm*, int *n*)

Randomly permutes the perm array of size n. Assumes that perm[] is already filled with the elements to be permuted.

Definition at line 110 of file RNG.cpp.

4.12 inc/Swap.h File Reference

Classes

- class Swap

4.13 inc/SwapEnds.h File Reference

Classes

- class SwapEnds

4.14 inc/Sweep.h File Reference

Classes

- class Sweep

Defines

- #define VRPH_UNUSED 0
- #define VRPH_ADDED 1
- #define VRPH_INTERIOR 2

4.14.1 Define Documentation

4.14.1.1 #define VRPH_ADDED 1

Definition at line 17 of file Sweep.h.

4.14.1.2 #define VRPH_INTERIOR 2

Definition at line 18 of file Sweep.h.

4.14.1.3 #define VRPH_UNUSED 0

Definition at line 16 of file Sweep.h.

4.15 inc/ThreeOpt.h File Reference

Classes

- class ThreeOpt

4.16 inc/ThreePointMove.h File Reference

Classes

- class ThreePointMove

4.17 inc/TwoOpt.h File Reference

Classes

- class TwoOpt

4.18 inc/TwoPointMove.h File Reference

Classes

- class TwoPointMove

4.19 inc/VRP.h File Reference

Classes

- class VRP

4.20 inc/VRPConfig.h File Reference

Defines

- #define EPS_EXE "epstopdf"
- #define RESEED_RNG 0
- #define FORBID_TINY_MOVES 1

4.20.1 Define Documentation

4.20.1.1 #define EPS_EXE "epstopdf"

Definition at line 13 of file VRPConfig.h.

4.20.1.2 #define FORBID_TINY_MOVES 1

Definition at line 27 of file VRPConfig.h.

4.20.1.3 #define RESEED_RNG 0

Definition at line 23 of file VRPConfig.h.

4.21 inc/VRPDebug.h File Reference

Defines

- #define FIXED_DEBUG 0
- #define SEARCH_DEBUG 0
- #define VRPH_TABU_DEBUG 0
- #define BLOAT_DEBUG 0
- #define WAREHOUSE_DEBUG 0
- #define VERIFY_ALL 0
- #define CW_DEBUG 0
- #define CLEAN_DEBUG 0
- #define Q_DEBUG 0
- #define Q_VERIFY 0 + VERIFY_ALL
- #define OP_VERIFY 0
- #define STRING_DEBUG 0
- #define STRING_VERIFY 0 + VERIFY_ALL
- #define OPM_VERIFY 0 + VERIFY_ALL
- #define OPM_DEBUG 0
- #define OR_VERIFY 0 + VERIFY_ALL
- #define OR_DEBUG 0
- #define POSTSERT_VERIFY 0 + VERIFY_ALL
- #define POSTSERT_DEBUG 0
- #define PRESERT_VERIFY 0 + VERIFY_ALL
- #define PRESERT_DEBUG 0
- #define FLIP_DEBUG 0
- #define FLIP_VERIFY 0 + VERIFY_ALL
- #define SWAP_ENDS_DEBUG 0
- #define SWAP_ENDS_VERIFY 0 + VERIFY_ALL
- #define SWAP_DEBUG 0
- #define SWAP_VERIFY 0 + VERIFY_ALL
- #define REVERSE_DEBUG 0
- #define REVERSE_VERIFY 0 + VERIFY_ALL
- #define SWAP_VERIFY 0 + VERIFY_ALL
- #define SWAP_DEBUG 0
- #define CONCATENATE_DEBUG 0
- #define CONCATENATE_VERIFY 0 + VERIFY_ALL
- #define TPM_DEBUG 0
- #define TPM_VERIFY 0 + VERIFY_ALL
- #define TWO_OPT_DEBUG 0
- #define TWO_OPT_VERIFY 0 + VERIFY_ALL
- #define THREE_OPT_DEBUG 0

- #define THREE_OPT_VERIFY 0 + VERIFY_ALL
- #define CROSS_EXCHANGE_DEBUG 0
- #define CROSS_EXCHANGE_VERIFY 0 + VERIFY_ALL
- #define NEIGHBOR_DEBUG 0
- #define TSPLIB_DEBUG 0

Functions

- void report_error (const char *format,...)

4.21.1 Define Documentation

4.21.1.1 #define BLOAT_DEBUG 0

Definition at line 19 of file VRPDebug.h.

4.21.1.2 #define CLEAN_DEBUG 0

Definition at line 25 of file VRPDebug.h.

4.21.1.3 #define CONCATENATE_DEBUG 0

Definition at line 62 of file VRPDebug.h.

4.21.1.4 #define CONCATENATE_VERIFY 0 + VERIFY_ALL

Definition at line 63 of file VRPDebug.h.

4.21.1.5 #define CROSS_EXCHANGE_DEBUG 0

Definition at line 75 of file VRPDebug.h.

4.21.1.6 #define CROSS_EXCHANGE_VERIFY 0 + VERIFY_ALL

Definition at line 76 of file VRPDebug.h.

4.21.1.7 #define CW_DEBUG 0

Definition at line 23 of file VRPDebug.h.

4.21.1.8 #define FIXED_DEBUG 0

Definition at line 16 of file VRPDebug.h.

4.21.1.9 #define FLIP_DEBUG 0

Definition at line 47 of file VRPDebug.h.

4.21.1.10 #define FLIP_VERIFY 0 + VERIFY_ALL

Definition at line 48 of file VRPDebug.h.

4.21.1.11 #define NEIGHBOR_DEBUG 0

Definition at line 79 of file VRPDebug.h.

4.21.1.12 #define OP_VERIFY 0

Definition at line 30 of file VRPDebug.h.

4.21.1.13 #define OPM_DEBUG 0

Definition at line 36 of file VRPDebug.h.

4.21.1.14 #define OPM_VERIFY 0 + VERIFY_ALL

Definition at line 35 of file VRPDebug.h.

4.21.1.15 #define OR_DEBUG 0

Definition at line 39 of file VRPDebug.h.

4.21.1.16 #define OR_VERIFY 0 + VERIFY_ALL

Definition at line 38 of file VRPDebug.h.

4.21.1.17 #define POSTSERT_DEBUG 0

Definition at line 42 of file VRPDebug.h.

4.21.1.18 #define POSTSERT_VERIFY 0 + VERIFY_ALL

Definition at line 41 of file VRPDebug.h.

4.21.1.19 #define PRESERT_DEBUG 0

Definition at line 45 of file VRPDebug.h.

4.21.1.20 #define PRESERT_VERIFY 0 + VERIFY_ALL

Definition at line 44 of file VRPDebug.h.

4.21.1.21 #define Q_DEBUG 0

Definition at line 27 of file VRPDebug.h.

4.21.1.22 #define Q_VERIFY 0 + VERIFY_ALL

Definition at line 28 of file VRPDebug.h.

4.21.1.23 #define REVERSE_DEBUG 0

Definition at line 56 of file VRPDebug.h.

4.21.1.24 #define REVERSE_VERIFY 0 + VERIFY_ALL

Definition at line 57 of file VRPDebug.h.

4.21.1.25 #define SEARCH_DEBUG 0

Definition at line 17 of file VRPDebug.h.

4.21.1.26 #define STRING_DEBUG 0

Definition at line 32 of file VRPDebug.h.

4.21.1.27 #define STRING_VERIFY 0 + VERIFY_ALL

Definition at line 33 of file VRPDebug.h.

4.21.1.28 #define SWAP_DEBUG 0

Definition at line 60 of file VRPDebug.h.

4.21.1.29 #define SWAP_DEBUG 0

Definition at line 60 of file VRPDebug.h.

4.21.1.30 #define SWAP_ENDS_DEBUG 0

Definition at line 50 of file VRPDebug.h.

4.21.1.31 #define SWAP_ENDS_VERIFY 0 + VERIFY_ALL

Definition at line 51 of file VRPDebug.h.

4.21.1.32 #define SWAP_VERIFY 0 + VERIFY_ALL

Definition at line 59 of file VRPDebug.h.

4.21.1.33 #define SWAP_VERIFY 0 + VERIFY_ALL

Definition at line 59 of file VRPDebug.h.

4.21.1.34 #define THREE_OPT_DEBUG 0

Definition at line 72 of file VRPDebug.h.

4.21.1.35 #define THREE_OPT_VERIFY 0 + VERIFY_ALL

Definition at line 73 of file VRPDebug.h.

4.21.1.36 #define TPM_DEBUG 0

Definition at line 66 of file VRPDebug.h.

4.21.1.37 #define TPM_VERIFY 0 + VERIFY_ALL

Definition at line 67 of file VRPDebug.h.

4.21.1.38 #define TSPLIB_DEBUG 0

Definition at line 80 of file VRPDebug.h.

4.21.1.39 #define TWO_OPT_DEBUG 0

Definition at line 69 of file VRPDebug.h.

4.21.1.40 #define TWO_OPT_VERIFY 0 + VERIFY_ALL

Definition at line 70 of file VRPDebug.h.

4.21.1.41 #define VERIFY_ALL 0

Definition at line 21 of file VRPDebug.h.

4.21.1.42 #define VRPH_TABU_DEBUG 0

Definition at line 18 of file VRPDebug.h.

4.21.1.43 #define WAREHOUSE_DEBUG 0

Definition at line 20 of file VRPDebug.h.

4.21.2 Function Documentation

4.21.2.1 void report_error (const char * *format*, ...)

Prints the message and function name to stderr and terminates the program.

Definition at line 298 of file VRPDebug.cpp.

4.22 inc/VRPGenerator.h File Reference

Functions

- void generate_li_vrp (int A, int B, int Q, int L, const char *outfile)

4.22.1 Function Documentation

4.22.1.1 void generate_li_vrp (int *A*, int *B*, int *Q*, int *L*, const char * *outfile*)

Generates a TSPLIB-formatted VRP file given the parameters using the generator of Li, et al., 2005.

Definition at line 12 of file VRPGenerator.cpp.

4.23 inc/VRPH.h File Reference

```
#include "RNG.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <memory.h>
#include <assert.h>
#include "VRPDebug.h"
#include "VRPHeuristic.h"
#include "VRPUtils.h"
#include "VRPNode.h"
#include "VRPRoute.h"
#include "VRPMove.h"
#include "VRPSolution.h"
#include "VRPTabuList.h"
#include "VRP.h"
#include "Postsert.h"
#include "Presert.h"
#include "Concatenate.h"
#include "SwapEnds.h"
#include "Flip.h"
#include "Swap.h"
#include "MoveString.h"
#include "OnePointMove.h"
#include "TwoPointMove.h"
#include "TwoOpt.h"
#include "ClarkeWright.h"
#include "Sweep.h"
```

```
#include "OrOpt.h"
#include "ThreeOpt.h"
#include "CrossExchange.h"
#include "VRPGenerator.h"
#include "ThreePointMove.h"
```

Defines

- #define VRPH_TSP 1
- #define VRPH_CVRP 2
- #define VRPH_FUNCTION 1
- #define VRPH_UPPER_ROW 2
- #define VRPH_FULL_MATRIX 3
- #define VRPH_LOWER_ROW 4
- #define VRPH_UPPER_DIAG_ROW 5
- #define VRPH_LOWER_DIAG_ROW 6
- #define VRPH_TWOD_COORDS 2
- #define VRPH_THREED_COORDS 3
- #define VRPH_EXPLICIT 0
- #define VRPH_EUC_2D 1
- #define VRPH_EUC_3D 2
- #define VRPH_MAX_2D 3
- #define VRPH_MAX_3D 4
- #define VRPH_MAN_2D 5
- #define VRPH_MAN_3D 6
- #define VRPH_CEIL_2D 7
- #define VRPH_GEO 8
- #define VRPH_EXACT_2D 9
- #define VRPH_MIN(X, Y) ((X) < (Y) ? (X) : (Y))
- #define VRPH_MAX(X, Y) ((X) < (Y) ? (Y) : (X))
- #define VRPH_ABS(a) (((a) < 0) ? -(a) : (a))
- #define VRPH_RANDOM_SEARCH 1
- #define VRPH_REGRET_SEARCH 2
- #define VRPH_EPS_EXE "epstopdf"
- #define VRPH_BLACK 0
- #define VRPH_RED 1
- #define VRPH_YELLOW 2
- #define VRPH_GREEN 3
- #define VRPH_AQUA 4
- #define VRPH_PINK 5
- #define VRPH_WHEAT 6

- #define VRPH_GRAY 7
- #define VRPH_BROWN 8
- #define VRPH_BLUE 9
- #define VRPH_VIOLET 10
- #define VRPH_CYAN 11
- #define VRPH_TURQUOISE 12
- #define VRPH_MAGENTA 13
- #define VRPH_SALMON 14
- #define VRPH_WHITE 15
- #define VRPH_DEFAULT_PLOT 0
- #define VRPH_BLACK_AND_WHITE 1
- #define VRPH_COLOR 2
- #define VRPH_BOXED 4
- #define VRPH_NO_TITLE 8
- #define VRPH_BARE_BONES 16
- #define VRPH_NO_POINTS 32
- #define VRPH_NO_DEPOT_EDGES 64
- #define VRPH_WEIGHTED 128
- #define VRPH_ADD_ENTROPY 0
- #define VRPH_FORBID_TINY_MOVES 1
- #define VRPH_MAX_NUM_LAMBDAS 100
- #define VRPH_STRING_SIZE 200
- #define VRPH_DEPOT 0
- #define VRPH_PI 3.14159265358979323846264
- #define VRPH_RRR 6378.3888
- #define VRP_INFINITY (1<<30)
- #define VRP_INFEASIBLE VRP_INFINITY
- #define VRPH_EPSILON .00001
- #define VRPH_DEFAULT_DEVIATION .01
- #define VRPH_MAX_NUM_ROUTES 10000
- #define VRPH_LI_PERTURB 0
- #define VRPH_OSMAN_PERTURB 1
- #define VRPH_MAX_SERVICE_DAYS 10

Functions

- void VRPH_version ()

4.23.1 Define Documentation

4.23.1.1 #define VRP_INFEASIBLE VRP_INFINITY

Definition at line 108 of file VRPH.h.

4.23.1.2 #define VRP_INFINITY (1<<30)

Definition at line 107 of file VRPH.h.

4.23.1.3 #define VRPH_ABS(a) (((a) < 0) ? -(a) : (a))

Definition at line 51 of file VRPH.h.

4.23.1.4 #define VRPH_ADD_ENTROPY 0

Definition at line 98 of file VRPH.h.

4.23.1.5 #define VRPH_AQUA 4

Definition at line 70 of file VRPH.h.

4.23.1.6 #define VRPH_BARE_BONES 16

Definition at line 89 of file VRPH.h.

4.23.1.7 #define VRPH_BLACK 0

Definition at line 66 of file VRPH.h.

4.23.1.8 #define VRPH_BLACK_AND_WHITE 1

Definition at line 85 of file VRPH.h.

4.23.1.9 #define VRPH_BLUE 9

Definition at line 75 of file VRPH.h.

4.23.1.10 #define VRPH_BOXED 4

Definition at line 87 of file VRPH.h.

4.23.1.11 #define VRPH_BROWN 8

Definition at line 74 of file VRPH.h.

4.23.1.12 #define VRPH_CEIL_2D 7

Definition at line 44 of file VRPH.h.

4.23.1.13 #define VRPH_COLOR 2

Definition at line 86 of file VRPH.h.

4.23.1.14 #define VRPH_CVRP 2

Definition at line 22 of file VRPH.h.

4.23.1.15 #define VRPH_CYAN 11

Definition at line 77 of file VRPH.h.

4.23.1.16 #define VRPH_DEFAULT_DEVIATION .01

Definition at line 110 of file VRPH.h.

4.23.1.17 #define VRPH_DEFAULT_PLOT 0

Definition at line 84 of file VRPH.h.

4.23.1.18 #define VRPH_DEPOT 0

Definition at line 104 of file VRPH.h.

4.23.1.19 #define VRPH_EPS_EXE "epstopdf"

Definition at line 58 of file VRPH.h.

4.23.1.20 #define VRPH_EPSILON .00001

Definition at line 109 of file VRPH.h.

4.23.1.21 #define VRPH_EUC_2D 1

Definition at line 38 of file VRPH.h.

4.23.1.22 #define VRPH_EUC_3D 2

Definition at line 39 of file VRPH.h.

4.23.1.23 #define VRPH_EXACT_2D 9

Definition at line 46 of file VRPH.h.

4.23.1.24 #define VRPH_EXPLICIT 0

Definition at line 37 of file VRPH.h.

4.23.1.25 #define VRPH_FORBID_TINY_MOVES 1

Definition at line 101 of file VRPH.h.

4.23.1.26 #define VRPH_FULL_MATRIX 3

Definition at line 27 of file VRPH.h.

4.23.1.27 #define VRPH_FUNCTION 1

Definition at line 25 of file VRPH.h.

4.23.1.28 #define VRPH_GEO 8

Definition at line 45 of file VRPH.h.

4.23.1.29 #define VRPH_GRAY 7

Definition at line 73 of file VRPH.h.

4.23.1.30 #define VRPH_GREEN 3

Definition at line 69 of file VRPH.h.

4.23.1.31 #define VRPH_LI_PERTURB 0

Definition at line 113 of file VRPH.h.

4.23.1.32 #define VRPH_LOWER_DIAG_ROW 6

Definition at line 30 of file VRPH.h.

4.23.1.33 #define VRPH_LOWER_ROW 4

Definition at line 28 of file VRPH.h.

4.23.1.34 #define VRPH_MAGENTA 13

Definition at line 79 of file VRPH.h.

4.23.1.35 #define VRPH_MAN_2D 5

Definition at line 42 of file VRPH.h.

4.23.1.36 #define VRPH_MAN_3D 6

Definition at line 43 of file VRPH.h.

4.23.1.37 #define VRPH_MAX(X, Y) ((X) < (Y) ? (Y) : (X))

Definition at line 50 of file VRPH.h.

4.23.1.38 #define VRPH_MAX_2D 3

Definition at line 40 of file VRPH.h.

4.23.1.39 #define VRPH_MAX_3D 4

Definition at line 41 of file VRPH.h.

4.23.1.40 #define VRPH_MAX_NUM_LAMBDAS 100

Definition at line 102 of file VRPH.h.

4.23.1.41 #define VRPH_MAX_NUM_ROUTES 10000

Definition at line 111 of file VRPH.h.

4.23.1.42 #define VRPH_MAX_SERVICE_DAYS 10

Definition at line 117 of file VRPH.h.

4.23.1.43 #define VRPH_MIN(X, Y) ((X) < (Y) ? (X) : (Y))

Definition at line 49 of file VRPH.h.

4.23.1.44 #define VRPH_NO_DEPOT_EDGES 64

Definition at line 91 of file VRPH.h.

4.23.1.45 #define VRPH_NO_POINTS 32

Definition at line 90 of file VRPH.h.

4.23.1.46 #define VRPH_NO_TITLE 8

Definition at line 88 of file VRPH.h.

4.23.1.47 #define VRPH_OSMAN_PERTURB 1

Definition at line 115 of file VRPH.h.

4.23.1.48 #define VRPH_PI 3.1415926535879323846264

Definition at line 105 of file VRPH.h.

4.23.1.49 #define VRPH_PINK 5

Definition at line 71 of file VRPH.h.

4.23.1.50 #define VRPH_RANDOM_SEARCH 1

Definition at line 54 of file VRPH.h.

4.23.1.51 #define VRPH_RED 1

Definition at line 67 of file VRPH.h.

4.23.1.52 #define VRPH_REGRET_SEARCH 2

Definition at line 55 of file VRPH.h.

4.23.1.53 #define VRPH_RRR 6378.3888

Definition at line 106 of file VRPH.h.

4.23.1.54 #define VRPH_SALMON 14

Definition at line 80 of file VRPH.h.

4.23.1.55 #define VRPH_STRING_SIZE 200

Definition at line 103 of file VRPH.h.

4.23.1.56 #define VRPH_THREED_COORDS 3

Definition at line 34 of file VRPH.h.

4.23.1.57 #define VRPH_TSP 1

Definition at line 21 of file VRPH.h.

4.23.1.58 #define VRPH_TURQUOISE 12

Definition at line 78 of file VRPH.h.

4.23.1.59 #define VRPH_TWOD_COORDS 2

Definition at line 33 of file VRPH.h.

4.23.1.60 #define VRPH_UPPER_DIAG_ROW 5

Definition at line 29 of file VRPH.h.

4.23.1.61 #define VRPH_UPPER_ROW 2

Definition at line 26 of file VRPH.h.

4.23.1.62 #define VRPH_VIOLET 10

Definition at line 76 of file VRPH.h.

4.23.1.63 #define VRPH_WEIGHTED 128

Definition at line 92 of file VRPH.h.

4.23.1.64 #define VRPH_WHEAT 6

Definition at line 72 of file VRPH.h.

4.23.1.65 #define VRPH_WHITE 15

Definition at line 81 of file VRPH.h.

4.23.1.66 #define VRPH_YELLOW 2

Definition at line 68 of file VRPH.h.

4.23.2 Function Documentation

4.23.2.1 void VRPH_version ()

Definition at line 16 of file VRP.cpp.

4.24 inc/VRPHeuristic.h File Reference

Defines

- #define VRPH_DOWNHILL 1
- #define VRPH_RECORD_TO_RECORD (1<<1)
- #define VRPH_SIMULATED_ANNEALING (1<<2)
- #define VRPH_FIRST_ACCEPT (1<<3)
- #define VRPH_BEST_ACCEPT (1<<4)
- #define VRPH_LI_ACCEPT (1<<5)
- #define VRPH_INTER_ROUTE_ONLY (1<<6)
- #define VRPH_INTRA_ROUTE_ONLY (1<<7)
- #define VRPH_USE_NEIGHBOR_LIST (1<<8)
- #define VRPH_FREE (1<<9)
- #define VRPH_BALANCED (1<<10)
- #define VRPH_FORWARD (1<<11)
- #define VRPH_BACKWARD (1<<12)
- #define VRPH_RANDOMIZED (1<<13)
- #define VRPH_SAVINGS_ONLY (1<<14)
- #define VRPH_MINIMIZE_NUM_ROUTES (1<<15)
- #define VRPH_FIXED_EDGES (1<<17)
- #define VRPH_ALLOW_INFEASIBLE (1<<18)
- #define VRPH_NO_NEW_ROUTE (1<<19)
- #define VRPH_TABU (1<<20)
- #define ONE_POINT_MOVE (1<<21)
- #define TWO_POINT_MOVE (1<<22)
- #define TWO_OPT (1<<23)
- #define OR_OPT (1<<24)
- #define THREE_OPT (1<<25)
- #define CROSS_EXCHANGE (1<<26)
- #define THREE_POINT_MOVE (1<<27)
- #define KITCHEN_SINK (1<<28)
- #define ALL_HEURISTICS (1<<20)|(1<<21)|(1<<22)|(1<<23)|(1<<24)|(1<<25)|(1<<26)|(1<<27
- #define NUM_HEURISTICS 7
- #define ONE_POINT_MOVE_INDEX 0
- #define TWO_POINT_MOVE_INDEX 1
- #define TWO_OPT_INDEX 2
- #define OR_OPT_INDEX 3
- #define THREE_OPT_INDEX 4
- #define CROSS_EXCHANGE_INDEX 5
- #define THREE_POINT_MOVE_INDEX 6
- #define PRESERT 1

- #define POSTSERT 2
- #define CONCATENATE 3
- #define SWAP_ENDS 4
- #define FLIP 5
- #define MOVE_STRING 6
- #define SWAP 7

4.24.1 Define Documentation

4.24.1.1 #define ALL_HEURISTICS (1<<20)|(1<<21)|(1<<22)|(1<<23)|(1<<24)|(1<<25)|(1<<26)|(1<<27)

Definition at line 48 of file VRPHeuristic.h.

4.24.1.2 #define CONCATENATE 3

Definition at line 68 of file VRPHeuristic.h.

4.24.1.3 #define CROSS_EXCHANGE (1<<26)

Definition at line 44 of file VRPHeuristic.h.

4.24.1.4 #define CROSS_EXCHANGE_INDEX 5

Definition at line 60 of file VRPHeuristic.h.

4.24.1.5 #define FLIP 5

Definition at line 70 of file VRPHeuristic.h.

4.24.1.6 #define KITCHEN_SINK (1<<28)

Definition at line 46 of file VRPHeuristic.h.

4.24.1.7 #define MOVE_STRING 6

Definition at line 71 of file VRPHeuristic.h.

4.24.1.8 #define NUM_HEURISTICS 7

Definition at line 53 of file VRPHeuristic.h.

4.24.1.9 #define ONE_POINT_MOVE (1<<21)

Definition at line 39 of file VRPHeuristic.h.

4.24.1.10 #define ONE_POINT_MOVE_INDEX 0

Definition at line 55 of file VRPHeuristic.h.

4.24.1.11 #define OR_OPT (1<<24)

Definition at line 42 of file VRPHeuristic.h.

4.24.1.12 #define OR_OPT_INDEX 3

Definition at line 58 of file VRPHeuristic.h.

4.24.1.13 #define POSTSERT 2

Definition at line 67 of file VRPHeuristic.h.

4.24.1.14 #define PRESERT 1

Definition at line 66 of file VRPHeuristic.h.

4.24.1.15 #define SWAP 7

Definition at line 72 of file VRPHeuristic.h.

4.24.1.16 #define SWAP_ENDS 4

Definition at line 69 of file VRPHeuristic.h.

4.24.1.17 #define THREE_OPT (1<<25)

Definition at line 43 of file VRPHeuristic.h.

4.24.1.18 #define THREE_OPT_INDEX 4

Definition at line 59 of file VRPHeuristic.h.

4.24.1.19 #define THREE_POINT_MOVE (1<<27)

Definition at line 45 of file VRPHeuristic.h.

4.24.1.20 #define THREE_POINT_MOVE_INDEX 6

Definition at line 61 of file VRPHeuristic.h.

4.24.1.21 #define TWO_OPT (1<<23)

Definition at line 41 of file VRPHeuristic.h.

4.24.1.22 #define TWO_OPT_INDEX 2

Definition at line 57 of file VRPHeuristic.h.

4.24.1.23 #define TWO_POINT_MOVE (1<<22)

Definition at line 40 of file VRPHeuristic.h.

4.24.1.24 #define TWO_POINT_MOVE_INDEX 1

Definition at line 56 of file VRPHeuristic.h.

4.24.1.25 #define VRPH_ALLOW_INFEASIBLE (1<<18)

Definition at line 34 of file VRPHeuristic.h.

4.24.1.26 #define VRPH_BACKWARD (1<<12)

Definition at line 29 of file VRPHeuristic.h.

4.24.1.27 #define VRPH_BALANCED (1<<10)

Definition at line 27 of file VRPHeuristic.h.

4.24.1.28 #define VRPH_BEST_ACCEPT (1<<4)

Definition at line 21 of file VRPHeuristic.h.

4.24.1.29 #define VRPH_DOWNHILL 1

Definition at line 17 of file VRPHeuristic.h.

4.24.1.30 #define VRPH_FIRST_ACCEPT (1<<3)

Definition at line 20 of file VRPHeuristic.h.

4.24.1.31 #define VRPH_FIXED_EDGES (1<<17)

Definition at line 33 of file VRPHeuristic.h.

4.24.1.32 #define VRPH_FORWARD (1<<11)

Definition at line 28 of file VRPHeuristic.h.

4.24.1.33 #define VRPH_FREE (1<<9)

Definition at line 26 of file VRPHeuristic.h.

4.24.1.34 #define VRPH_INTER_ROUTE_ONLY (1<<6)

Definition at line 23 of file VRPHeuristic.h.

4.24.1.35 #define VRPH_INTRA_ROUTE_ONLY (1<<7)

Definition at line 24 of file VRPHeuristic.h.

4.24.1.36 #define VRPH_LI_ACCEPT (1<<5)

Definition at line 22 of file VRPHeuristic.h.

4.24.1.37 #define VRPH_MINIMIZE_NUM_ROUTES (1<<15)

Definition at line 32 of file VRPHeuristic.h.

4.24.1.38 #define VRPH_NO_NEW_ROUTE (1<<19)

Definition at line 35 of file VRPHeuristic.h.

4.24.1.39 #define VRPH_RANDOMIZED (1<<13)

Definition at line 30 of file VRPHeuristic.h.

4.24.1.40 #define VRPH_RECORD_TO_RECORD (1<<1)

Definition at line 18 of file VRPHeuristic.h.

4.24.1.41 #define VRPH_SAVINGS_ONLY (1<<14)

Definition at line 31 of file VRPHeuristic.h.

4.24.1.42 #define VRPH_SIMULATED_ANNEALING (1<<2)

Definition at line 19 of file VRPHeuristic.h.

4.24.1.43 #define VRPH_TABU (1<<20)

Definition at line 36 of file VRPHeuristic.h.

4.24.1.44 #define VRPH_USE_NEIGHBOR_LIST (1<<8)

Definition at line 25 of file VRPHeuristic.h.

4.25 inc/VRPMove.h File Reference

Classes

- class VRPMove

Defines

- #define MAX_AFFECTED_ROUTES 3
- #define MAX_ARGUMENTS 15

4.25.1 Define Documentation

4.25.1.1 #define MAX_AFFECTED_ROUTES 3

Definition at line 15 of file VRPMove.h.

4.25.1.2 #define MAX_ARGUMENTS 15

Definition at line 16 of file VRPMove.h.

4.26 inc/VRPNode.h File Reference

Classes

- class VRPNode

Defines

- #define VRPTW 0
- #define MAX_NEIGHBORLIST_SIZE 75

4.26.1 Define Documentation

4.26.1.1 #define MAX_NEIGHBORLIST_SIZE 75

Definition at line 17 of file VRPNode.h.

4.26.1.2 #define VRPTW 0

Definition at line 16 of file VRPNode.h.

4.27 inc/VRPRoute.h File Reference

Classes

- class VRPRoute
- class VRPRouteWarehouse

Defines

- #define MAX_NEIGHBORING_ROUTES 5
- #define DUPLICATE_ROUTE 0
- #define OVERWRITTEN_ROUTE 1
- #define ADDED_ROUTE 2
- #define BETTER_ROUTE 3

4.27.1 Define Documentation

4.27.1.1 #define ADDED_ROUTE 2

Definition at line 20 of file VRPRoute.h.

4.27.1.2 #define BETTER_ROUTE 3

Definition at line 21 of file VRPRoute.h.

4.27.1.3 #define DUPLICATE_ROUTE 0

Definition at line 18 of file VRPRoute.h.

4.27.1.4 #define MAX_NEIGHBORING_ROUTES 5

Definition at line 17 of file VRPRoute.h.

4.27.1.5 #define OVERWRITTEN_ROUTE 1

Definition at line 19 of file VRPRoute.h.

4.28 inc/VRPSolution.h File Reference

Classes

- class VRPSolution
- class VRPSolutionWarehouse

4.29 inc/VRPTabuList.h File Reference

Classes

- class VRPTabuList

Defines

- #define NUM_VRPH_TABU_ROUTES 50

4.29.1 Define Documentation

4.29.1.1 #define NUM_VRPH_TABU_ROUTES 50

Definition at line 16 of file VRPTabuList.h.

4.30 inc/VRPUtils.h File Reference

Classes

- struct htable_entry
- struct int_int
- struct double_int
- class VRPSavingsElement
- class VRPNeighborElement
- class VRPViolation
- class VRPSeedElement
- class VRPNeighborhood
- struct VRPSegment

Defines

- #define MAX_FILES 20000
- #define MAX_FILENAME_LENGTH 40
- #define NUM_ELITE_SOLUTIONS 200
- #define MAX_NUM_COLS 10000
- #define NUM_ENTRIES 8
- #define MAX_VRPH_TABU_LIST_SIZE 50
- #define HASH_TABLE_SIZE (1<<18)
- #define SALT_1 0
- #define SALT_2 11

Functions

- double VRPDistance (int type, double x1, double y1, double x2, double y2)
- int VRPDistanceCompare (const void *a, const void *b)
- int VRPIntCompare (const void *a, const void *b)
- int VRPSavingsCompare (const void *a, const void *b)
- int VRPNeighborCompare (const void *a, const void *b)
- int VRPAlphaCompare (const void *a, const void *b)
- int double_int_compare (const void *a, const void *b)
- int int_int_compare (const void *a, const void *b)
- int VRPSolutionCompare (const void *a, const void *b)
- int VRPCheckTSPLIBString (char *s)
- int VRPGetDimension (char *filename)
- int VRPGetNumDays (char *filename)

4.30.1 Define Documentation

4.30.1.1 #define HASH_TABLE_SIZE (1<<18)

Definition at line 26 of file VRPUtils.h.

4.30.1.2 #define MAX_FILENAME_LENGTH 40

Definition at line 18 of file VRPUtils.h.

4.30.1.3 #define MAX_FILES 20000

Definition at line 17 of file VRPUtils.h.

4.30.1.4 #define MAX_NUM_COLS 10000

Definition at line 22 of file VRPUtils.h.

4.30.1.5 #define MAX_VRPH_TABU_LIST_SIZE 50

Definition at line 24 of file VRPUtils.h.

4.30.1.6 #define NUM_ELITE_SOLUTIONS 200

Definition at line 21 of file VRPUtils.h.

4.30.1.7 #define NUM_ENTRIES 8

Definition at line 23 of file VRPUtils.h.

4.30.1.8 #define SALT_1 0

Definition at line 27 of file VRPUtils.h.

4.30.1.9 #define SALT_2 11

Definition at line 28 of file VRPUtils.h.

4.30.2 Function Documentation

4.30.2.1 int double_int_compare (const void * *a*, const void * *b*)

Compares two double_int's using the double field.

Definition at line 107 of file VRPUtils.cpp.

4.30.2.2 int int_int_compare (const void * *a*, const void * *b*)

Compares two int_int's using the j field.

Definition at line 126 of file VRPUtils.cpp.

4.30.2.3 int VRPAlphaCompare (const void * *a*, const void * *b*)

Compares two strings and sorts alphabetically.

Definition at line 212 of file VRPUtils.cpp.

4.30.2.4 int VRPCheckTSPLIBString (char * *s*)

Determines whether or not a given string in an input file is a supported TSPLIB string. Returns the reference number for the string if supported, and 0 otherwise.

Definition at line 172 of file VRPTSPLib.cpp.

4.30.2.5 double VRPDistance (int *type*, double *x1*, double *y1*, double *x2*, double *y2*)

Distance function for 2D problems.

Definition at line 15 of file VRPUtils.cpp.

4.30.2.6 int VRPDistanceCompare (const void * *a*, const void * *b*)

Compares two doubles.

Definition at line 78 of file VRPUtils.cpp.

4.30.2.7 int VRPGetDimension (char * *filename*)

Open up filename (assumed to be in TSPLIB format) and get the dimension of the problem, scanning for the string "DIMENSION" and makes sure that the "EOF" string is also found.

Definition at line 56 of file VRPTSPLib.cpp.

4.30.2.8 int VRPGetNumDays (char * *filename*)

Open up filename (assumed to be in TSPLIB format) and get the dimension of the problem, scanning for the string "NUM_DAYS". If the string is not found, then we assume it is a typical 1-day problem.

Definition at line 126 of file VRPTSPLib.cpp.

4.30.2.9 int VRPIntCompare (const void * *a*, const void * *b*)

Compares two int's.

Definition at line 97 of file VRPUtils.cpp.

4.30.2.10 int VRPNeighborCompare (const void * *a*, const void * *b*)

Compares two VRPNeighborElements using the val field.

Definition at line 168 of file VRPUtils.cpp.

4.30.2.11 int VRPSavingsCompare (const void * *a*, const void * *b*)

Compares two VRPSavingsElement's using the savings field.

Definition at line 144 of file VRPUtils.cpp.

4.30.2.12 int VRPSolutionCompare (const void * *a*, const void * *b*)

Compares two VRPSolution's using the obj field.

Definition at line 192 of file VRPUtils.cpp.

4.31 src/apps/SYMPHONY_vrp_main.c File Reference

```
#include "symphony.h"
#include "sym_master.h"
#include "vrp_types.h"
#include "vrp_const.h"
#include "VRPH.h"
```

Defines

- #define COMPILING_FOR_MASTER
- #define USE_VRPH 1
- #define NUM_VRPH_SOLUTIONS 10

Functions

- int vrp_test (sym_environment *env, int argc, char **argv)
- int main (int argc, char **argv)

4.31.1 Define Documentation

4.31.1.1 #define COMPILING_FOR_MASTER

Definition at line 21 of file SYMPHONY_vrp_main.c.

4.31.1.2 #define NUM_VRPH_SOLUTIONS 10

Definition at line 24 of file SYMPHONY_vrp_main.c.

4.31.1.3 #define USE_VRPH 1

Definition at line 22 of file SYMPHONY_vrp_main.c.

4.31.2 Function Documentation

4.31.2.1 int main (int *argc*, char ** *argv*)

Definition at line 75 of file SYMPHONY_vrp_main.c.

4.31.2.2 int vrp_test (sym_environment * *env*, int *argc*, char ** *argv*)

Definition at line 173 of file SYMPHONY_vrp_main.c.

4.32 src/apps/vrp_ej.cpp File Reference

```
#include "VRPH.h"
```

Defines

- #define RANDOM 0
- #define REGRET 1

Functions

- int main (int argc, char *argv[ ])

4.32.1 Define Documentation

4.32.1.1 #define RANDOM 0

Definition at line 15 of file vrp_ej.cpp.

4.32.1.2 #define REGRET 1

Definition at line 16 of file vrp_ej.cpp.

4.32.2 Function Documentation

4.32.2.1 int main (int *argc*, char * *argv*[ ])

A main() routine to test the procedures and performance of various routines for ejecting and injecting groups of nodes using two strategies.

Definition at line 18 of file vrp_ej.cpp.

4.33 src/apps/vrp_glpk_sp.cpp File Reference

```
#include "OsiGlpkSolverInterface.hpp"
#include "CoinPackedMatrix.hpp"
#include "CoinPackedVector.hpp"
#include "VRPH.h"
```

Defines

- #define MAX_ROUTES 50000

Functions

- void OSI_recover_route (int id, int **orderings, VRPRoute *r)
- void OSI_recover_solution (OsiSolverInterface *si, int **orderings, int *sol_-buff)
- void OSI_add_route (OsiSolverInterface *si, VRP *V, VRPRoute *r, int id, int **orderings)
- int main (int argc, char **argv)

Variables

- int max_columns
- int num_cols_to_delete
- bool verbose
- time_t heur_time
- time_t mip_time

4.33.1 Define Documentation

4.33.1.1 #define MAX_ROUTES 50000

Definition at line 23 of file vrp_glpk_sp.cpp.

4.33.2 Function Documentation

4.33.2.1 int main (int *argc*, char ** *argv*)

Definition at line 225 of file vrp_glpk_sp.cpp.

4.33.2.2 void OSI_add_route (OsiSolverInterface * *si*, VRP * *V*, VRPRoute * *r*, int *id*, int ** *orderings*)

Adds a column/route to the current set partitioning problem. The column is assigned the name of id, which is converted to a string.

Definition at line 108 of file vrp_glpk_sp.cpp.

4.33.2.3 void OSI_recover_route (int *id*, int ** *orderings*, VRPRoute * *r*)

Populates the route r with the information from orderings[i]. Computes the number of customers and the hash values of the route.

Definition at line 38 of file vrp_glpk_sp.cpp.

4.33.2.4 void OSI_recover_solution (OsiSolverInterface * *si*, int ** *orderings*, int * *sol_buff*)

Extracts the solution from the current set partitioning instance and places it in sol_buff[]. We need to access the column/variable names since these hold the orderings of the routes.

Definition at line 64 of file vrp_glpk_sp.cpp.

4.33.3 Variable Documentation

4.33.3.1 time_t heur_time

Definition at line 36 of file vrp_glpk_sp.cpp.

4.33.3.2 int max_columns

Definition at line 32 of file vrp_glpk_sp.cpp.

4.33.3.3 time_t mip_time

Definition at line 36 of file vrp_glpk_sp.cpp.

4.33.3.4 int num_cols_to_delete

Definition at line 33 of file vrp_glpk_sp.cpp.

4.33.3.5 bool verbose

Definition at line 35 of file vrp_glpk_sp.cpp.

4.34 src/apps/vrp_initial.cpp File Reference

```
#include "VRPH.h"
```

Defines

- #define RANDOM 0
- #define REGRET 1

Functions

- int main (int argc, char *argv[ ])

4.34.1 Define Documentation

4.34.1.1 #define RANDOM 0

Definition at line 15 of file vrp_initial.cpp.

4.34.1.2 #define REGRET 1

Definition at line 16 of file vrp_initial.cpp.

4.34.2 Function Documentation

4.34.2.1 int main (int *argc*, char * *argv*[ ])

A main() routine to illustrate usage of Clarke Wright and Sweep methods to construct an initial solution.

Definition at line 18 of file vrp_initial.cpp.

4.35 src/apps/vrp_plotter.cpp File Reference

```
#include "VRPH.h"
```

Functions

- int main (int argc, char *argv[ ])

4.35.1 Function Documentation

4.35.1.1 int main (int *argc*, char * *argv*[ ])

This is the main() routine that demonstrates how to import solutions and plot them.

Definition at line 16 of file vrp_plotter.cpp.

4.36 src/apps/vrp_rtr.cpp File Reference

```
#include "VRPH.h"
#include <time.h>
```

Functions

- int main (int argc, char *argv[ ])

4.36.1 Function Documentation

4.36.1.1 int main (int *argc*, char * *argv*[ ])

This is the main() routine to construct a command line tool for solving VRP's using the record-to-record travel algorithm.

Definition at line 16 of file vrp_rtr.cpp.

4.37 src/apps/vrp_sa.cpp File Reference

```
#include "VRPH.h"
```

Functions

- int main (int argc, char *argv[ ])

4.37.1 Function Documentation

4.37.1.1 int main (int *argc*, char * *argv*[ ])

This is a main() routine that uses Simulated Annealing to generate VRP solutions.

Definition at line 15 of file vrp_sa.cpp.

4.38 src/ClarkeWright.cpp File Reference

```
#include "VRPH.h"
```

4.39 src/Concatenate.cpp File Reference

```
#include "VRPH.h"
```

4.40 src/CrossExchange.cpp File Reference

```
#include "VRPH.h"
```

4.41 src/Flip.cpp File Reference

```
#include "VRPH.h"
```

4.42 src/MoveString.cpp File Reference

```
#include "VRPH.h"
```

4.43 src/OnePointMove.cpp File Reference

```
#include "VRPH.h"
```

4.44 src/OrOpt.cpp File Reference

```
#include "VRPH.h"
```

4.45 src/Postsert.cpp File Reference

```
#include "VRPH.h"
```

4.46 src/Presert.cpp File Reference

```
#include "VRPH.h"
```

4.47 src/RNG.cpp File Reference

```
#include "VRPH.h"
```

Defines

- #define MODULUS 2147483647
- #define MULT 16807

Functions

- double lcgrand (int stream)
- void random_permutation (int *perm, int n)

Variables

- static long long zrng [ ]

4.47.1 Define Documentation

4.47.1.1 #define MODULUS 2147483647

Definition at line 14 of file RNG.cpp.

4.47.1.2 #define MULT 16807

Definition at line 15 of file RNG.cpp.

4.47.2 Function Documentation

4.47.2.1 double lcgrand (int *stream*)

A run of the mill LCG.

Definition at line 79 of file RNG.cpp.

4.47.2.2 void random_permutation (int * *perm*, int *n*)

Randomly permutes the perm array of size n. Assumes that perm[] is already filled with the elements to be permuted.

Definition at line 110 of file RNG.cpp.

4.47.3 Variable Documentation

4.47.3.1 long long zrng[ ] [static]

Definition at line 19 of file RNG.cpp.

4.48 src/Swap.cpp File Reference

```
#include "VRPH.h"
```

4.49 src/SwapEnds.cpp File Reference

```
#include "VRPH.h"
```

4.50 src/Sweep.cpp File Reference

```
#include "VRPH.h"
```

Classes

- struct sweep_node

4.51 src/ThreeOpt.cpp File Reference

```
#include "VRPH.h"
```

4.52 src/ThreePointMove.cpp File Reference

```
#include "VRPH.h"
```

4.53 src/TwoOpt.cpp File Reference

```
#include "VRPH.h"
```

4.54 src/TwoPointMove.cpp File Reference

```
#include "VRPH.h"
```

4.55 src/VRP.cpp File Reference

```
#include "VRPH.h"
```

Defines

- #define VRPH_MAX_CYCLES 500

Functions

- void VRPH_version ()

4.55.1 Define Documentation

4.55.1.1 #define VRPH_MAX_CYCLES 500

Definition at line 14 of file VRP.cpp.

4.55.2 Function Documentation

4.55.2.1 void VRPH_version ()

Definition at line 16 of file VRP.cpp.

4.56 src/VRPDebug.cpp File Reference

```
#include "VRPH.h"
#include <stdarg.h>
```

Functions

- void report_error (const char *format,...)

4.56.1 Function Documentation

4.56.1.1 void report_error (const char * *format*, ...)

Prints the message and function name to stderr and terminates the program.

Definition at line 298 of file VRPDebug.cpp.

4.57 src/VRPGenerator.cpp File Reference

```
#include "VRPH.h"
```

Functions

- void generate_li_vrp (int A, int B, int Q, int L, const char *outfile)

4.57.1 Function Documentation

4.57.1.1 void generate_li_vrp (int *A*, int *B*, int *Q*, int *L*, const char * *outfile*)

Generates a TSPLIB-formatted VRP file given the parameters using the generator of Li, et al., 2005.

Definition at line 12 of file VRPGenerator.cpp.

4.58 src/VRPGraphics.cpp File Reference

```
#include "VRPH.h"
```

4.59 src/VRPIO.cpp File Reference

```
#include "VRPH.h"
```

4.60 src/VRPMove.cpp File Reference

```
#include "VRPH.h"
```

4.61 src/VRPNode.cpp File Reference

```
#include "VRPH.h"
```

4.62 src/VRPRoute.cpp File Reference

```
#include "VRPH.h"
#include "randvals.h"
```

4.63 src/VRPSolution.cpp File Reference

```
#include "VRPH.h"
#include "randvals.h"
```

4.64 src/VRPSolvers.cpp File Reference

```
#include "VRPH.h"
```

4.65 src/VRPTabuList.cpp File Reference

```
#include "VRPH.h"
```

4.66 src/VRPTSPLib.cpp File Reference

```
#include "VRPH.h"
```

Functions

- int VRPGetDimension (char *filename)
- int VRPGetNumDays (char *filename)
- int VRPCheckTSPLIBString (char *s)

Variables

- const char * SupportedTSPLIBStrings [ ]
- const int SL [ ]
- const int NumSupportedTSPLIBStrings = 25
- const char * UnsupportedTSPLIBStrings [ ]
- const int NumUnsupportedTSPLIBStrings = 20

4.66.1 Function Documentation

4.66.1.1 int VRPCheckTSPLIBString (char * *s*)

Determines whether or not a given string in an input file is a supported TSPLIB string. Returns the reference number for the string if supported, and 0 otherwise.

Definition at line 172 of file VRPTSPLib.cpp.

4.66.1.2 int VRPGetDimension (char * *filename*)

Open up filename (assumed to be in TSPLIB format) and get the dimension of the problem, scanning for the string "DIMENSION" and makes sure that the "EOF" string is also found.

Definition at line 56 of file VRPTSPLib.cpp.

4.66.1.3 int VRPGetNumDays (char * *filename*)

Open up filename (assumed to be in TSPLIB format) and get the dimension of the problem, scanning for the string "NUM_DAYS". If the string is not found, then we assume it is a typical 1-day problem.

Definition at line 126 of file VRPTSPLib.cpp.

4.66.2 Variable Documentation

4.66.2.1 const int NumSupportedTSPLIBStrings = 25

Definition at line 42 of file VRPTSPLib.cpp.

4.66.2.2 const int NumUnsupportedTSPLIBStrings = 20

Definition at line 54 of file VRPTSPLib.cpp.

4.66.2.3 const int SL[ ]

Initial value:

```
{4,4,10,9,
                8,8,18,16,
                14,3,18,13,
                13,14,12,8,
                8,16,19,7,
                20,12,17,10,
                13}
```

Definition at line 33 of file VRPTSPLib.cpp.

4.66.2.4 const char* SupportedTSPLIBStrings[ ]

Initial value:

```
{
    "NAME","TYPE","BEST_KNOWN","DIMENSION",

    "CAPACITY","DISTANCE","EDGE_WEIGHT_FORMAT","EDGE_WEIGHT_TYPE",

    "NODE_COORD_TYPE","EOF","NODE_COORD_SECTION","DEPOT_SECTION",

    "DEMAND_SECTION","EDGE_WEIGHT_SECTION","SERVICE_TIME","cxd",

    "NUM_DAYS","SVC_TIME_SECTION","TIME_WINDOW_SECTION","COMMENT",

    "DISPLAY_DATA_SECTION","TWOD_DISPLAY","DISPLAY_DATA_TYPE","NO_DISPLAY",

    "COORD_DISPLAY"}
```

Definition at line 16 of file VRPTSPLib.cpp.

4.66.2.5 const char* UnsupportedTSPLIBStrings[ ]

Initial value:

```
    {
    "HCP","ATSP","SOP","TOUR","ATT","XRAY1","XRAY2","SPECIAL",
    "LOWER_ROW",
    "LOWER_DIAG_ROW","UPPER_COL","LOWER_COL","UPPER_DIAG_COL",
    "LOWER_DIAG_COL","EDGE_LIST","ADJ_LIST","NO_COORDS",
    "EDGE_DATA_SECTION",
    "TOUR_SECTION"
}
```

Definition at line 45 of file VRPTSPLib.cpp.

4.67 src/VRPUtils.cpp File Reference

```
#include "VRPH.h"
```

Functions

- double VRPDistance (int type, double x1, double y1, double x2, double y2)
- int VRPDistanceCompare (const void *a, const void *b)
- int VRPIntCompare (const void *a, const void *b)
- int double_int_compare (const void *a, const void *b)
- int int_int_compare (const void *a, const void *b)
- int VRPSavingsCompare (const void *a, const void *b)
- int VRPNeighborCompare (const void *a, const void *b)
- int VRPSolutionCompare (const void *a, const void *b)
- int VRPAlphaCompare (const void *a, const void *b)
- int VRPRouteCompare (const void *a, const void *b)

4.67.1 Function Documentation

4.67.1.1 int double_int_compare (const void * *a*, const void * *b*)

Compares two double_int's using the double field.

Definition at line 107 of file VRPUtils.cpp.

4.67.1.2 int int_int_compare (const void * *a*, const void * *b*)

Compares two int_int's using the j field.

Definition at line 126 of file VRPUtils.cpp.

4.67.1.3 int VRPAlphaCompare (const void * *a*, const void * *b*)

Compares two strings and sorts alphabetically.

Definition at line 212 of file VRPUtils.cpp.

4.67.1.4 double VRPDistance (int *type*, double *x1*, double *y1*, double *x2*, double *y2*)

Distance function for 2D problems.

Definition at line 15 of file VRPUtils.cpp.

4.67.1.5 int VRPDistanceCompare (const void * a, const void * b)

Compares two doubles.

Definition at line 78 of file VRPUtils.cpp.

4.67.1.6 int VRPIntCompare (const void * a, const void * b)

Compares two int's.

Definition at line 97 of file VRPUtils.cpp.

4.67.1.7 int VRPNeighborCompare (const void * a, const void * b)

Compares two VRPNeighborElements using the val field.

Definition at line 168 of file VRPUtils.cpp.

4.67.1.8 int VRPRouteCompare (const void * a, const void * b)

Compares two VRPRoutes using the length field.

Definition at line 221 of file VRPUtils.cpp.

4.67.1.9 int VRPSavingsCompare (const void * a, const void * b)

Compares two VRPSavingsElement's using the savings field.

Definition at line 144 of file VRPUtils.cpp.

4.67.1.10 int VRPSolutionCompare (const void * a, const void * b)

Compares two VRPSolution's using the obj field.

Definition at line 192 of file VRPUtils.cpp.